VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Geolokační systém uvnitř budovy založený na technologii Bluetooth

Indoor Geolocation System Based on Bluetooth Technology

2018

Bc. Martin Sadový

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Sadový**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Geolokační systém uvnitř budovy založený na technologii Bluetooth
Indoor Geolocation System Based on Bluetooth Technology

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem diplomové práce je vytvoření geolokačního systému fungujícího uvnitř budovy založeného na technologii Bluetooth. Systém bude schopen určit polohu zařízení v budově vybavené sítí vysílačů Bluetooth. Dále bude systém schopen navrhnout optimální cestu ke konkrétnímu místu v budově.

1. Stručně popište výstupy semestrálního projektu na který tato diplomová práce navazuje.
2. Navrhněte vhodný hardware (pouze modul pro Bluetooth) pro vysílače Bluetooth umístěné v budově. Bude-li to nutné vytvořte software pro tento modul, tak aby ho bylo možné využít pro lokalizační systém.
3. Popište vámi navržený princip geolokace a věnujte se možnostem zpřesnění výsledků s využitím různých filtrů (kalman, particle, neuronové sítě, případně další) případně s využitím dalších senzorů (akcelerometr, gyroskop).
4. Vytvořte aplikaci pro OS Android, která bude spolupracovat s Bluetooth vysílači umístěnými v budově a bude schopna určit polohu zařízení a zobrazit jí na mapových podkladech.
5. Aplikace bude obsahovat funkci vyhledávání optimální trasy mezi dvěma body.
6. Systém otestujte a popište jeho chování i v nestandardních situacích (například hluchá místa bez signálu, přechod mezi jednotlivými patry budovy, aj.).

Seznam doporučené odborné literatury:

[1] TOWNSEND, Kevin., Robert. DAVIDSON, AKIBA. a Carles. CUFI. Getting started with Bluetooth low energy: tools and techniques for low-power networking. Revised First Edition. ISBN 978-1491949511.
[2] DARWIN, Ian. Android Cookbook: Problems and Solutions for Android Developers. Second Edition. ISBN 978-1449374433.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Velice rád bych poděkoval svému vedoucímu Ing. Davidu Seidlovi Ph.D. především při finálním dokončování této práce a Ing. Davidu Ježkovi Ph.D. za konzultaci okolo neuronových sítí.

Abstrakt

Diplomová práce se zabývá geolokací uvnitř budovy. V práci se lze dozvědět o využití filtrů (průměrový filtr, mediánový filtr, Kalmanův Filtr, Particle Filter a Neuronové sítě) pro zpřesnění výsledků geolokačního algoritmu. V práci je popsáno jakým způsobem lze implementovat Bluetooth Low Energy majáček na platformě ESP32. Práce se taktéž zaobírá možnosti mapování vnitřních prostor pomocí Simple Indoor Tagging z ekosystému OpenStreetMap a vyhledávání cesty mezi dvěma body.

Klíčová slova: Bluetooth Low Energy, ESP32, Android, geolokace, vnitřní prostory, Android, Particle filter, umělá neuronová síť, Kotlin, Reaktivní programování, vyhledávání trasy, Eddystone, majáček

Abstract

The Master's thesis deals with the problem of indoor geolocation. The thesis describes ways how to use filters (Mean, Median, Kalman, Particle, and Artificial neural network) to increase accuracy of indoor geolocation. The thesis describes practical implementation of an indoor geolocation system based on Bluetooth Low Energy beacons, which are based on ESP32. The thesis describes indoor mapping with Simple Indoor Tagging by OpenStreetMap and how to deal with pathfinding inside a mapped building.

Key Words: Bluetooth Low Energy, ESP32, android, geolocation, indoor, Android, Particle filter, artificial neural network, Kotlin, ReactiveX, pathfinding, Eddystone, beacon

Obsah

Seznam použitých zkratek a symbolů

AD	–	Advertising data
AES	–	Advanced Encryption Standard
API	–	Application Programming Interface
ASCII	–	American Standard Code for Information Interchange
BID	–	Instance ID
BLE	–	Bluetooth Low Energy
BSSID	–	Basic Service Set Identifier
BTS	–	Základnová převodní stanice
CSS	–	Kaskádové styly
CSV	–	Comma-separated values
DRSS	–	Differential received signal strength
ECC	–	Kryptografie nad eliptickými křivkami
GAP	–	Generic Access Profile
GPS	–	Globální polohový systém
HW	–	Hardware
I^2C	–	Inter-Integrated Circuit
JOSM	–	Java OpenStreetMap editor
JSON	–	JavaScript Object Notation
KF	–	Kalmanův filter
MAC	–	Media Access Control
NFC	–	Near Field Communication
NID	–	Namespace ID
NN	–	Neuronová síť
OSM	–	Open Street Map
PDU	–	Protokolová datová jednotka
PF	–	Particle filter
POI	–	Point of interest
RSA	–	Kryptografická asymetrická šifra
RSS	–	Received Signal Strength
RSSI	–	Received Signal Strength Indicator
SDK	–	Software development kit
SHA-2	–	Secure Hash Algorithm 2
SPI	–	Serial Peripheral Interface
SSID	–	Service Set Identifier
SoC	–	Systém na čipu
TCP	–	Transmission Control Protocol

TDoA	–	Time difference of arrival
ToA	–	Time of arrival
UART	–	Universal asynchronous receiver-transmitter
USB	–	Universal Serial Bus
UUID	–	Universally unique identifier
VHCI	–	Virtual Host Controller Interface
Wi-Fi	–	Wireless Fidelity
WGS84	–	World Geodetic System 1984

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Dnešní mobilní zařízení disponují technologiemi pro zjištění geografické polohy (dále geolokace). Mezi tyto technologie patří globální satelitní systémy, jako je například GPS, GLONASS či lokální satelitní systémy jako je BeiDou[1] v asiopacifickém regionu. Nevýhodou satelitních systémů je závislost na přímé či částečné viditelnosti družice z mobilního zařízení. Nejsou proto určené pro geolokaci v podzemí, uvnitř budov či skrytých místech. Globální satelitní systémy při přímé viditelnosti poskytují přesnost v jednotkách metrů a jedná se tedy o vcelku přesný systém pro určení polohy.

Jako doplňkovou službu lze využít seznam stanic (BTS) mobilních operátorů, ke kterým je uživatel připojen. Podle tohoto seznamu lze zjistit přibližnou polohu mobilního zařízení. Příkladem využití je koordinace zdravotnických služeb a mobilních operátorů pro lokalizaci postižené osoby[2].

Další možností geolokace je využití mobilních datových služeb pro připojení k databázi o všudypřítomných zařízeních (např. Wi-Fi hotspoty) a získat takto polohu z této databáze. Příkladem této varianty je Android a služby Google[3], kde operační systém ostatních uživatelů této platformy využívá přesných geolokačních technik (např. satelitní systém) a získává z okolních zařízení identifikátory, ať už je to SSID a BSSID z Wi-Fi sítí či data o BTS mobilních operátorů, které si anonymizovaně ukládá do své databáze a ty následně poskytuje ostatním účastníkům. Výsledkem dotazu může být poloha s přesností od jednotek metrů až po jednotky kilometrů. Tato databáze se i nadále rozšiřuje.

1.1 Geolokační služby pro vnitřní prostory

V obchodních domech, nemocnicích a jiných institucích můžeme naleznout informační stánky, které informují o trase zákazníka k hledanému místu. Tento způsob navigace však nemusí být pro některé jedince jednoduchý úkol - problémem je zapamatovatelnost celé trasy.

V takovémto případě se nabízejí chytrá zařízení, která se stala pravou rukou při každodenních činnostech, ať už je to komunikace nebo vyhledání dopravního spojení. Chytrá zařízení, jako je například mobilní telefon, dokáží nahradit papírovou mapu. Navíc vyhledají optimální trasu, kterou lze mít stále zobrazenou i při pohybu uživatele.

Geolokační služby pro vnitřní prostory momentálně postrádají standardizované řešení a existují proto pouze proprietární řešení, která jsou obvykle zpoplatněná.

[1] Od roku 2020 má být BeiDou globální. `http://www.globaltimes.cn/content/1073811.shtml`

[2] `http://radyvnouzi.cz/co-delat/tisnove-volani/`

[3] `https://developers.google.com/maps/documentation/geolocation/intro`

2 Shrnutí semestrální práce

2.1 Úvod

V roce 2016 jsem vedl s Ing. David Seidl, Ph.D. diskusi o tom, že se plánuje nasazení nového přístupového systému na univerzitě VŠB-TUO. A protože nasazení nového systému přináší nové možnosti, které s aktuálním systémem nejsou možné, je dobré být připraven na možné rozšíření před samotnou realizací nového systému. Jednou z vedlejších vlastností tohoto systému je možnost geolokace uvnitř budov Technické univerzity Ostrava.

2.2 Volba technologií pro vysílací stanice

Při výzkumu bylo zapotřebí zvolit bezdrátovou technologii, která musí být široce podporovaná mobilními zařízeními uživatelů, je cenově dostupná a nejlépe také již integrovaná v procesoru (resp. SoC) přístupové stanice. Tyto parametry splňují pouze dvě technologie. Jednou z nich je Wi-Fi, ta je bohužel nevhodná při hustém pokrytí vysílačů, kde jeden kanál má znatelně vyšší šířku frekvenčního pásma oproti druhé technologii, kterou byl nový standard *Bluetooth 4 Low Energy* (zkráceně BLE). BLE[4] je nejdostupnější variantou a je přímo přizpůsobená pro minimální odběr energie s nízkými náklady. Z těchto důvodu jsem zvolil technologii BLE.

Pro vytvoření vysílací stanice s BLE byl použit Raspberry Pi 3[5] jako nejdostupnější hardware. Jedná se o malý počítač postavený na architektuře ARM s integrovanou podporou BLE, díky čemuž bylo možné jednoduše vytvořit vysílací stanice.

2.3 Měření signálu a vzdálenosti

Pro získání polohy mobilní stanice se využívají lokalizační algoritmy, kde na vstupu bývají zpravidla vzdálenosti od vysílačů, které lze získat mnoha způsoby, ať už je to časovým rozdílem příchodu signálu (ToA, TDoA) nebo rozdílem mezi vysílací a naměřenou sílou signálu (RSS, DRSS). V případě technologie BLE a mobilních zařízení, která reprezentují mobilní telefony, je nutné využít techniky síly signálu.

Z dostupných dokumentací mobilních aplikací jsem se dozvěděl o funkci převádějící měřenou sílu signálu na vzdálenost, která byla následně použita při implementaci lokalizačních algoritmů.

Toto rozhodnutí se při závěrečných zhodnoceních ukázalo jako nevyhovující. Tato metoda předpokládá, že všechny vysílací stanice BLE mají stejnou charakteristiku vztahu síly signálu a vzdálenosti, kterou reprezentoval jeden doporučovaný výpočetní vzorec. Z tohoto důvodu lokalizační algoritmy poskytovaly velice špatné výsledky, a proto byl vytvořen vlastní algoritmus, který spočíval ve výpočtu váženého průměru[6] mezi vysílači.

[4] `https://en.wikipedia.org/wiki/Bluetooth_Low_Energy`
[5] `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`
[6] Bližší vysílač má vyšší váhu

2.4 Mobilní zařízení

Pro účely mobilní stanice byl zvolen operační systém Android od společnosti Google, jakožto nejrozšířenější systém, který společně s operačním systémem iOS od společnosti Apple pokrývá 99.9% trhu za rok 2017 [7]. Důležitým faktorem při výběru technologie BLE bylo také to, že je podporovaná zmíněnými platformami.

V této i předchozí práci je však kladen důraz na mobilní platformu Android, avšak s budoucím příslibem možnosti bezproblémového rozšíření i na platformu iOS.

Operační systém Android poskytuje vývojářům velmi široké rozhraní pro programování aplikací (API). První podpora BLE byla vystavena ve verzi 18 (tzv. API level 18), avšak ve verzi 21 došlo ke změnám rozhraní[8]. V té době byl podíl zařízení s verzí 18 větší než 30 procent. Letos je již tento podíl pouhých 10 procent a stále klesá[9]. Z tohoto důvodu aplikace implementovala obě verze rozhraní. Kromě zmíněné změny rozhraní pro BLE, došlo ve verzi 23 ke změně politiky oprávnění, kde již aplikace nemá garantované oprávnění schválené uživatelem při instalaci. Od verze Androidu 6 se po aplikaci vyžaduje, aby uživatele žádala o požadované oprávnění k daným API až při jejím využití, což se týká i zařízení BLE[10], které může být použito ke sledování uživatele a tím pádem data ze senzoru lze považovat za citlivé údaje[11].

Výsledná mobilní aplikace měla dvě základní komponenty: službu a uživatelské rozhraní. Služba se starala o sběr dat z BLE vysílačů a následnou lokalizaci. Uživatelské rozhraní se skládalo z reprezentace budovy a bodu, který reprezentoval lokalizovanou mobilní stanici.

Grafické zobrazování budovy je vytvořeno pomocí webové aplikace, která využívá knihovnu leaflet [12], se kterou byla využita také knihovna pro reprezentaci vnitřních budov. Knihovna pro zobrazení budovy již není aktivně vyvíjená. A proto bylo nutné knihovnu přizpůsobit pro poslední verzi knihovny leaflet, doprogramovat, opravit a udržovat další vlastnosti v rámci proměnlivé specifikace pro mapování vnitřních prostor budov (*indoor*) v OpenStreetMap.

[7] `www.statista.com` - Global market share held by smartphone operating systems from 2009 to 2017
[8] `https://developer.android.com/about/versions/android-5.0.html`
[9] `https://www.bidouille.org/misc/androidcharts`
[10] `https://developer.android.com/training/permissions/requesting.html`
[11] `https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner.html`
[12] `http://leafletjs.com/`

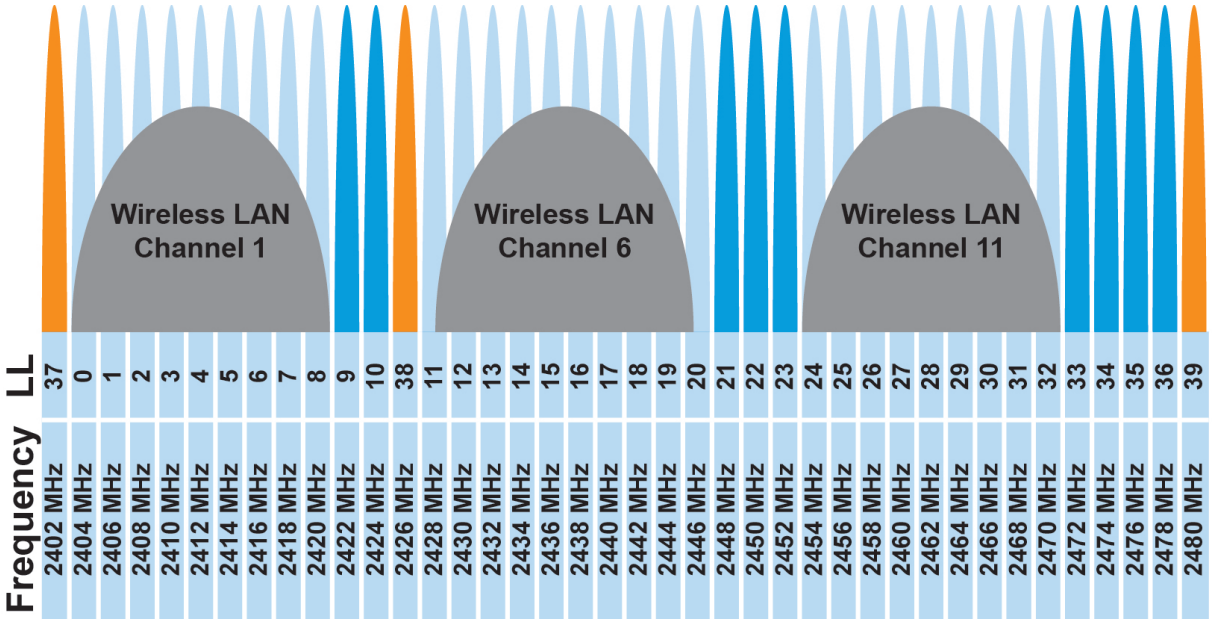
3 Vysílací stanice

Vysílací stanice využívá technologii Bluetooth 4 Low Energy (BLE)[13], jejíž klíčové vlastnosti pro potřeby vysílací stanice jsou nízká spotřeba, bezlicenční vysílací pásmo, nízké pořizovací náklady a podpora u mobilních stanic resp. chytrých telefonů. Nově pro účely vysílací stanice je použit modul ESP32[14].

3.1 Bluetooth Low Energy

V roce 2006 společnost Nokia uvedla technologii Wibree, kterou po dohodě se správcem technologie Bluetooth (Bluetooth SIG) přejmenovala na Bluetooth Low Energy. V roce 2010 došlo k začlenění této technologie do specifikace Bluetooth Core s pořadovým číslem čtyři [15].

Vzhledem k předpokladu malého množství přenesených dat a nízké spotřeby energie se předpokládá vysoká výdrž provozu zařízení na baterii. Díky tomu můžeme BLE nalézt v nositelných zařízeních, chytré domácnosti nebo senzorech v okolí [16]. V roce 2017 vznikla specifikace mesh sítě Bluetooth Mesh[17] postavená na technologii BLE, která je konkurentem pro technologii ZigBee.



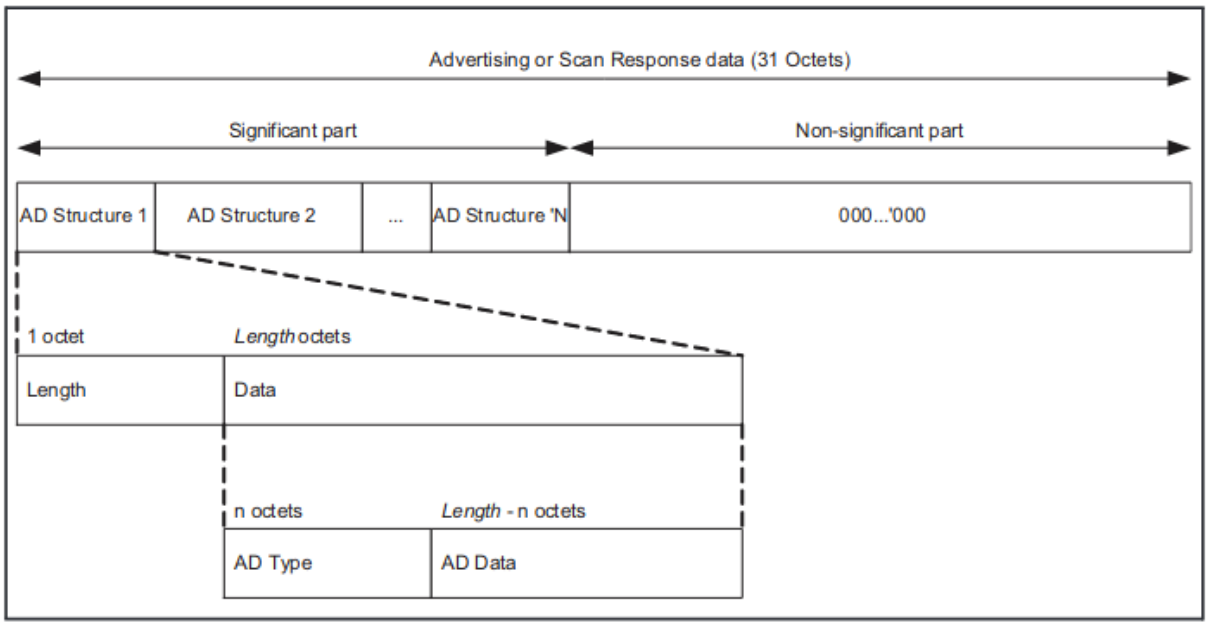
Obrázek 1: Kanály BLE se znázorněním nepřekrývajících se kanálů Wi-Fi. Zdroj: [11]

Bluetooth Low Energy využívá bezlicenční pásmo 2.4 Ghz, které je rozděleno do 40 kanálů po 2 Mhz (viz obrázek 1). Kanály 0 až 36 jsou datové kanály. Tyto kanály jsou využity v pří-

[13] https://www.bluetooth.com/specifications/bluetooth-core-specification
[14] https://www.espressif.com/en/products/hardware/esp32/overview
[15] https://en.wikipedia.org/wiki/Bluetooth_Low_Energy
[16] https://365tipu.cz/2017/09/06/tip883-co-je-to-bluetooth-le-ble-bluetooth-low-energy-bluetooth-smart-bluetooth-5/
[17] https://www.bluetooth.com/bluetooth-technology/topology-options/le-mesh

padě, že zařízení potřebuje předat data druhému zařízení. Zatímco kanály 37 až 39 jsou využity k tzv. "oznamování" (*advertising*), kde zařízení dává o sobě vědět a případně navazuje s druhým zařízením kontakt. Kanály jsou uspořádány tak, že pořadově vyšší kanál je na vyšší frekvenci s výjimkou kanálů 37 a 38, kde kanál 37 je před kanálem nula, 38 za kanálem 10. Tím se dospělo k tomu, že v případě využití technologie Wi-Fi na třech nepřekrývaných kanálech (1, 6 a 11) se kanály pro advertising překrývají s technologií Wi-Fi minimálně.

Základní datovou jednotkou BLE je *BLE Packet*. BLE Packet se skládá z preambule, přístupové adresy zařízení (nejedná se o MAC adresu), PDU a kontrolního součtu. PDU má minimální velikost 2 bajty. Ve verzích Bluetooth 4.0 a 4.1 dosahoval až 39 bajtů a od verze 4.2 může mít až 257 bajtů. PDU obsahuje vlastní hlavičku a "náklad"(*payload*). Důležitým prvkem v hlavičce je typ, který identifikuje zda zařízení jen vysílá (všem v okolí, či konkrétně) a zda se s ním dá navázat oboustranné spojení. [18] V payloadu je obsažena adresa vysílacího zařízení, která se stromovitě dělí. Adresa může být veřejná (výrobcem definovaná MAC adresa) nebo neveřejná. Neveřejná adresa může být náhodně vygenerovaná (statická) nebo případně skrytá *předsdíleným* klíčem. Důvodem je, aby zařízení nemohlo být dlouhodobě sledovatelné. [19]



Obrázek 2: Skládání AD v PDU v BLE paketu. Zdroj: [12]

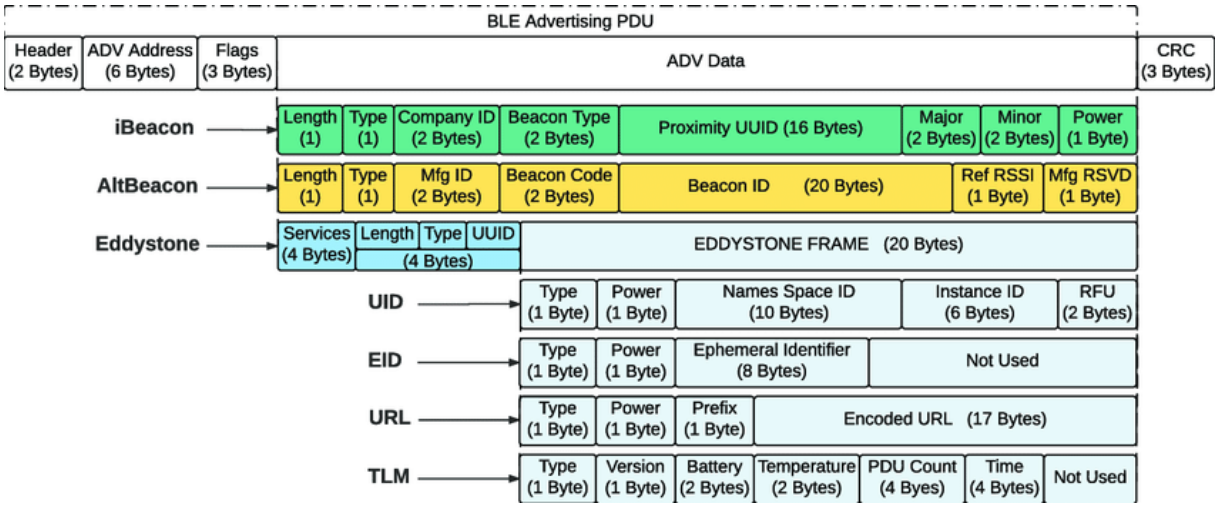
Za hlavičkou následuje část tzv. *Advertising Data* (jak ukazuje obrázek 2). Tato část se dělí na nezávislé bloky (dále AD). První bajt označuje velikost bloku, následuje identifikátor podle tabulky *Generic Access Profile* (GAP)[20]. Tento identifikátor může být z tabulky GAP

[18] `http://microchipdeveloper.com/wireless:ble-link-layer-packet-types`
[19] `http://microchipdeveloper.com/wireless:ble-link-layer-address`
[20] `https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile`, nebo Bluetooth Core 4 specifikace, Part C, oddíl 11 a 18

a tím pádem se jedná o data podle specifikace, nebo může nabývat nestandardizovaného kódu výrobce a data mohou mít libovolnou strukturu[21].

3.2 Beacon s podporou Eddystone™

Beacon, nebo-li majáček, je zařízení, které okolním zařízením dává o sobě znát. Díky tomu se uživatelé v okolí mohou dozvědět informace, jako jsou například aktuální odjezdy dopravních prostředků veřejné hromadné dopravy, a to právě na zastávce kde stojí, aniž by museli vybírat v aplikaci konkrétní zastávku.



Obrázek 3: Struktura dat ve vyslaném paketu majáčků různých společností. Zdroj: [13]

Mezi majáčky patří iBeacon od Apple, AltBeacon nebo Eddystone™ od Google. V případě Eddystone byl zakoupen vlastní UUID identifikátor [22] [23]. V roce 2014 Google vydal specifikaci UriBeacon, která se následně začlenila do specifikace Eddystone[24] jako Eddystone-URL s novým UUID. Tato specifikace zajišťuje čtyři druhy majáčků (viz obrázek 3).

Eddystone-UID vysílá identifikátor a změřenou hodnotu vysílacího výkonu ve vzdálenosti 0 m [25]. Identifikátor se skládá z *Namespace ID* (NID) a *Instance ID* (BID). NID je unikátní identifikátor společenství majáčků, který si každý tvůrce může vygenerovat UUID generátorem a následně spojí dva krajní segmenty[26] UUID. BID je následně identifikátorem jednoho konkrétního majáčku.

Eddystone-EID slouží jako utajený UID, který se mění a lze ho pouze unikátně identifikovat pomocí kryptografického výpočtu či pomocí vzdálené služby. Tento model se hodí v případě, že vývojář nechce, aby konkrétní majáček mohl být sledován.

[21] Této vlastnosti využívá iBeacon a AltBeacon
[22] `https://blog.bluetooth.com/advertising-works-part-1`
[23] `https://www.bluetooth.com/specifications/assigned-numbers/16-bit-uuids-for-members`
[24] `https://github.com/google/eddystone/blob/master/protocol-specification.md`
[25] Některé zdroje uvádí, že měřený výkon se měří ve vzdálenosti 1 m, nikoliv specifikace.
[26] Segment je v textové podobě UUID oddělený pomlčkou.

Eddystone-URL je kompaktní formát pro vysílání URL adresy. První bajt definuje vysílací výkon jako u UID. Druhý bajt definuje kombinaci protokolu (`http://` nebo `https://`) s nebo bez předpony `www.`. Následují tisknutelné znaky definované podle ASCII tabulky, kde netisknutelné znaky 0 až 13 jsou rezervované jako náhrada pro domény prvního řádu (případně s kombinací s lomítkem za doménou).

Eddystone-TLM přenáší informace o majáčku. Data mohou být vysílána zašifrovaně nebo čitelně. Tento PDU obsahuje stav baterie a teplotu majáčku, počet vyslaných PDU majáčkem a čas od zapnutí nebo restartu majáčku.

3.3 ESP32

ESP32 je malý SoC mikrokontrolér od firmy Espressif Systems. Modul bez antény měří 1.5 a 1.8 cm. Avšak samotný SoC měří 0.6 cm v obou délkách. Modul obsahuje 32 bitový dvou jádrový[27] Xtensa LX6 mikroprocesor s 520 KiB operační paměti a úsporný koprocesor. Hlavním důvodem výběru této jednotky je podpora Bluetooth 4.2 s BLE. Dále zde najdeme podporu Wi-Fi, sběrnice jako je SPI, I^2C, UART a Ethernet, taktéž nalezneme podporu kryptografických funkcí jako symetrická šifra AES, rodinu SHA-2 a asymetrickou kryptografii v podání RSA a ECC. Modul pracuje s napájecím napětím 3.3 V, avšak vývojový kit, lze připojit do USB, které pracuje na 5 V. USB taktéž funguje jako sériové rozhraní pro výstup programu či nahrávání nového programu a ladění[28]. Cena tohoto modulu se pohybuje kolem 100 Kč [29]

3.3.1 Implementace vysílací stanice

Společnost Espressif Systems pro ESP32 vydává velmi podrobnou dokumentaci a poskytuje podporu včetně bohatého SDK (ESP-IDF), kde nalezneme mnoho ukázek kódu. Z toho důvodu byla implementace majáčku pro tento modul velice snadná. Pro vytvoření BLE majáčku programátor využije rozhraní VHCI, kterým posílá pakety do Bluetooth kontroléru a ten následně odpovídá zpětným voláním. Kromě práce s mikrokontrolérem je možnost využít i operační systém reálného času FreeRTOS, kde programátor vytvoří úkol pro procesor.

Ke tvorbě majáčku pro operační systém Android postačuje, aby zařízení vysílalo alespoň jediný AD. V aplikaci lze následně identifikovat majáček pomocí MAC adresy. Bohužel existuje významný podíl iOS zařízení. API pro operační systém iOS neposkytuje MAC adresu[30] majáčku a z toho důvodu je potřeba zvolit jednu z podporovaných služeb, kterou je například Eddystone.

Jelikož ESP-IDF neobsahuje ukázky jak zprovoznit Eddystone majáček, bylo ho nutné naprogramovat. Prvním krokem bylo zjistit výsledný formát samotného obsahu vysílaného paketu a potřebné AD.

[27] Existuje model i s jedním jádrem

[28] `http://esp-idf.readthedocs.io/en/latest/api-guides/jtag-debugging/using-debugger.html`

[29] Cena z portálu Ebay.com - Únor 2018.

[30] `https://forums.developer.apple.com/thread/8442`

0x02	Velikost AD
0x01	Příznaky
0x06	LE General Discoverable Mode (0x04) | BR/EDR Not Supported (0x02)
0x03	Velikost AD
0x03	Seznam služeb podle 16 bitového UUID
0xAA	Eddystone UUID - LSB
0xFE	Eddystone UUID - MSB
3 až 23	Velikost AD včetně samotného Eddystone
0x16	Data spjaté se službou podle 16 bitového UUID
0xAA	Eddystone UUID - LSB
0xFE	Eddystone UUID - MSB
...	... Eddystone data v Big-Endian

Tabulka 1: Struktura AD v Eddystone packetu

V případě Eddystone se používají tři AD (viz tabulka 1):
První AD (horní box v tabulce 1) specifikuje vlastnosti daného zařízení. V tomto případě se jedná o zařízení, které je stále objevitelné[31], a nepodporuje Bluetooth Classic.
Druhý AD identifikuje službu, která je na zařízení provozována. V našem případě Google má registrovaný `0xFEAA` a používá ho pro Eddystone.
Třetí AD jsou data služby, která jsou provozovány na daném zařízení. V našem případě dostáváme Eddystone data, proto se zde taktéž vyskytuje UUID `0xFEAA`. Specifikace Bluetooth k zápisu více bajtové informace používá little endian, avšak Eddystone v tomto AD se zapisuje v Big-endian.

Vysílací stanice vysílá převážně Eddystone-UID. Velkou výhodou Eddystone je specifikace Eddystone-URL, díky které uživatelé v okolí mohou dostat informace do mobilního zařízení v podobě URL odkazu odkazující na užitečné informace. Bohužel Eddystone-UID již zabírá celý zbylý prostor PDU v paketu BLE. Z toho důvodu majáček vysílá i druhý paket s Eddystone-URL.

[31] Alternativou je omezený mód, kde zařízení vysílá advertising data pouze 30s.

4 Filtry

Měření reálných hodnot obsahuje mnoho nepřesností (odchylek). K částečnému odstranění těchto nepřesností se používají filtry.

Příkladem měřícího zařízení může být elektronický kompas v mobilním telefonu. Kompas měří směr magnetické indukce, kterou taktéž vyzařuje kterýkoliv spotřebič na elektrickou energii, přičemž sledovaná hodnota v reálném čase se tak může chovat lehce chaoticky. V tomto případě lze aplikovat filtr, zvaný dolní propust (*low-pass filter*), díky kterému je výsledná hodnota více stabilní.

Síla signálu (RSSI) může být velice kolísavá a pohybující se v určitém rozptylu hodnot. K tomuto účelu se dá použít průměrový filtr (*Mean filter*) a mediánový filtr (*Median filter* [1]), kde se z posledních naměřených hodnot vypočte průměr resp. medián.

Zmíněné filtry nejsou vhodné pro zpracování komplexnější množiny vstupních informací, jako například sledování polohy objektu. K tomuto účelu využíváme pokročilejší filtry, jako je Kalmanův filtr nebo Particle filter.

4.1 Kalmanův filtr

Kalmanův filtr (KF) je algoritmus, díky němuž lze odhadovat hodnotu měřené veličiny, která může být měřena zprostředkovaně a při měření je zatížená chybou (šumem)[32]. U KF mluvíme o dynamickém modelu, to znamená, že využíváme stav z předchozího měření, abychom předpověděli budoucí hodnotu měřené veličiny, na kterou v následujícím kroku aplikujeme korekci podle nově naměřené hodnoty.

Základní vlastností KF je pokus o řešení problému odhadu stavu (bodu) $x \in \mathbb{R}^n$ v diskrétním čase pomocí lineárních stochastických diferenciálních rovnic [16]:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{1a}$$

$$z_k = Hx_k + v_k \tag{1b}$$

kde index k značí číslo iterace (či diskrétní čas), $u \in \mathbb{R}^l$ je ovládací hodnota, z je naměřená hodnota, x je odhadovaný stav. Matice $A \in \mathbb{R}^{n \times n}$ značí změnu stavu z předchozí hodnoty na nynější bez začlenění šumu. Matice $B \in \mathbb{R}^{n \times l}$ je volitelná matice, tzv. kontrolní matice, která změní hodnotu vstupu (např. jiná veličina). Matice $H \in \mathbb{R}^{m \times n}$ je za účelem transformace stavu na výstupní hodnotu [33]. Matice A a H se mohou měnit v průběhu času.

[32] `https://www.youtube.com/watch?v=4OerJmPpkRg`

[33] Sekce 3.6. - `http://aircconline.com/ijcses/V8N1/8117ijcses01.pdf`

Dále jsou zde dvě náhodné veličiny, které jsou definované normálním rozdělením podle matic Q a R:

$$p(w) \sim N(0, Q) \tag{2a}$$

$$p(v) \sim N(0, R) \tag{2b}$$

kde Q je kovarianční matice šumu procesu, zatímco R je kovarianční matice šumu v měření. Jedná se o důležité faktory, které signifikantně ovlivňují výslednou kvalitu výstupů z KF. Pokud zvolíme větší Q znamená to, že máme menší jistotu v model, a tím pádem KF může výslednou hodnotu více opravovat podle naměřených hodnot. Obdobně, větší zvolené R znamená, že máme nižší jistotu v naměřených datech a filtr by tak měl méně opravovat stav podle naměřených dat [15].

Výsledný algoritmus se skládá ze dvou kroků **Predikce**:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{3a}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3b}$$

a **Korekce**

$$K_k = \frac{P_k^- H^T}{HP_k^- H^T + R} \tag{4a}$$

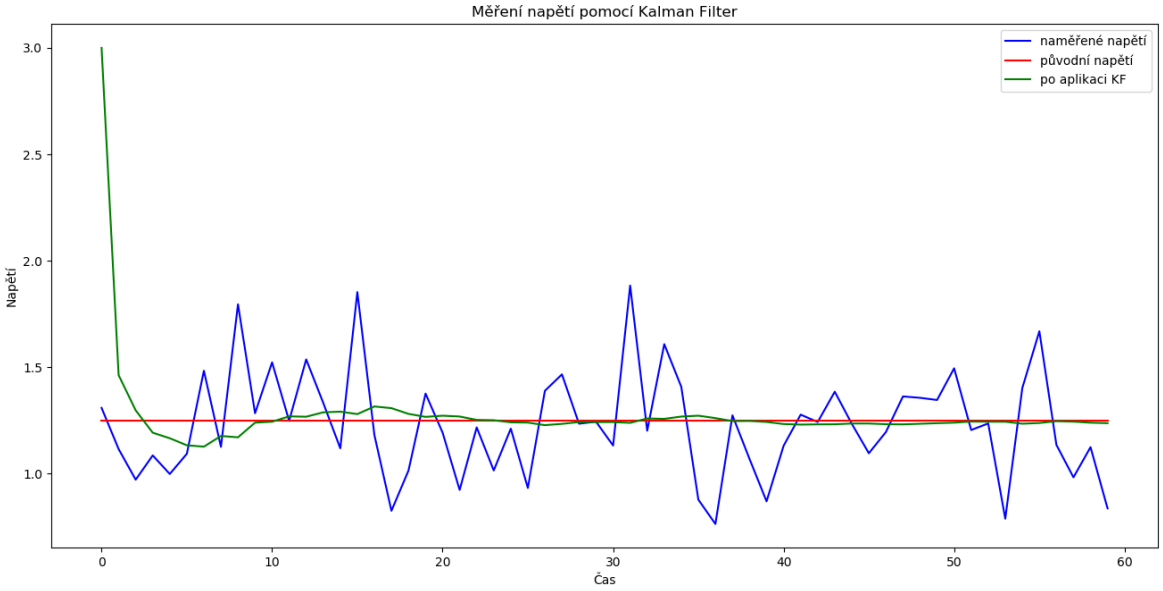
$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{4b}$$

$$P_k = (I - K_k H)P_k^- \tag{4c}$$

Postup algoritmu je následující:

1. V prvním inicializačním kroku je potřeba nastavit inicializační odhad stavu $\hat{x}_{k-1}$ a matici kovariance chyby P_{k-1}, včetně matic zmíněných v rovnicích (1) a (2).
2. V následujícím kroku je predikce, kde algoritmus provede podle předešlého stavu odhad nových výstupních hodnot (3a) a taktéž chybu predikce (3b).
3. V korekci se vypočte chybovost predikce (s chybou měření R) a vypočte se *Kalman gain* (4a), což je přírůstek k výstupní hodnotě rozdílu měření a predikce (4b).
4. Výsledný odhadovaný výstup v této iteraci je $\hat{x}_k$ (4b) s chybou P_k (4c). A pokračuje se bodem 2.

Při jednotlivém chybném měření se KF s pomocí P_k přikloní k predikci a takto eliminuje chybu měření.



Obrázek 4: Příklad simulace při použití Kalmanova filtru. Červená je reálná hodnota, modrá naměřená hodnota zatížená šumem a zelená je výstup z Kalmanova filtru

Příklad KF můžete vidět na obrázku 4, kde na vstupní model byla zvolena matice Q s jednou skalární hodnotou 0.00001. Díky čemuž KF dělá opravu měření ve prospěch modelu vůči naměřené hodnotě. [34]

Tento princip objevil Rudolf Emil Kálmán a jeho algoritmus se začal aktivně používat v letectví, navigačních systémech i dalších odvětvích. Kalmanův filtr řeší pouze lineární problémy. Pro nelineární problémy je vhodné použít Particle filter.

[34] Zdrojové kódy pro obrázek 4 `http://greg.czerniak.info/guides/kalman1/`

4.2 Particle filter

Particle filter (PF), taktéž známý pod názvem Sequential Monte Carlo (SMC), je genetický algoritmus, který vychází z rekurzivního Bayesova odhadu [35]. To v principu znamená, že nadcházející stav (x_k) je závislý bezprostředně na předchozím stavu (k_{k-1}), a není tak závislý na předešlých stavech (k_{k-2}, k_{k-3} atd.). Přepis algoritmu 1 [36]:

Algoritmus 1: Particle filter

1 algoritmus ParticleFilter;
  Input : N - počet částic
  X_{t-1} - set částic z předešlé iterace
  W_{t-1} - set vah částic z předchozí iterace
  u_t - pravděpodobnostní funkce pohybu
  z_t - pravděpodobnostní funkce výskytu
  Output: X_t - set částic z této interace
  W_t - set vah částic z této iterace
2 $X_t \leftarrow$ navzorkuj N částic podle vah W_{t-1} a pohybu u_t z částic X_{t-1} // sampling a pohyb;
3 **for** $1..N$ **do**
4 | $w_t^i = P(z_t|x_t^i)$ // vážení ;
5 **end**
6 $\mu = \sum_{i=0}^{N} w_t^i$;
7 **for** $1..N$ **do**
8 | $W_t^i = \frac{w_t^i}{\mu}$ // normalizace ;
9 **end**

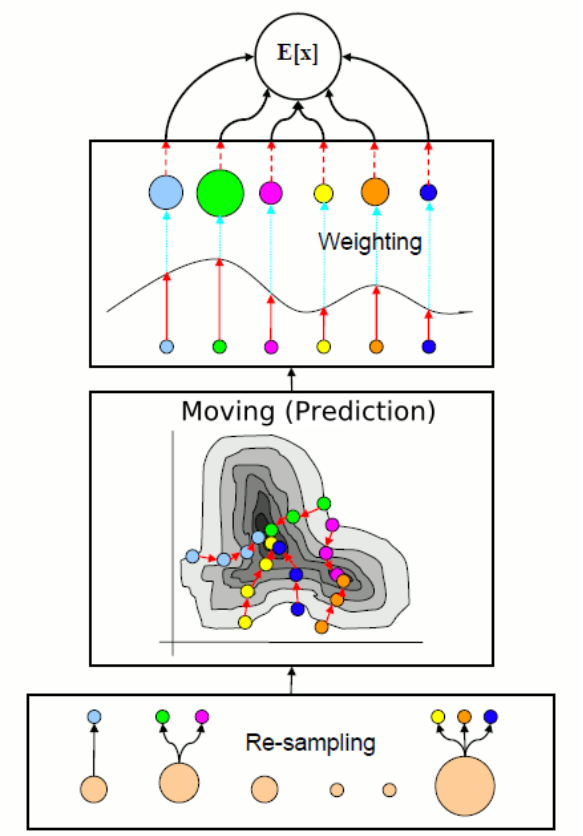
4.2.1 Popis algoritmu na příkladu využití

Z důvodu, že některé kroky ve formálním popisu algoritmu 1 jsou příliš zobecněné, tato kapitola se zaměří na ukázkový příklad využití.

Představme si letadlo, které letí nad známým členitým povrchem[37] (střídající se hory, údolí, roviny o různých nadmořských výškách), u tohoto letadla dokážeme měřit nadmořskou výšku a vzdálenost nad povrchem, avšak neznáme pozici, kde se letadlo nachází. A proto využijeme Particle filter.

Při první iteraci algoritmu po povrchu náhodně rozesejeme (*sampling*) (nejlépe rovnoměrně) N částic (*particles*). Tyto částice reprezentují s jakou pravděpodobností se letadlo na tomto místě vyskytuje, to znamená, že jedinec (částice) má určitou fitness z pohledu biologických algoritmů. Nyní všechny částice mají stejnou pravděpodobnost (v algoritmu dále označované jako váhu). V dalším kroku tyto částice ohodnotíme (*weighting*) a to tak, že zjistíme s jakou pravděpodobností se nachází letadlo právě na tomto místě. To znamená, že změříme nadmořskou

[35] https://en.wikipedia.org/wiki/Particle_filter
[36] https://www.youtube.com/watch?v=OM5iXrvUr_o
[37] https://www.youtube.com/watch?v=aUkBa1zMKv4



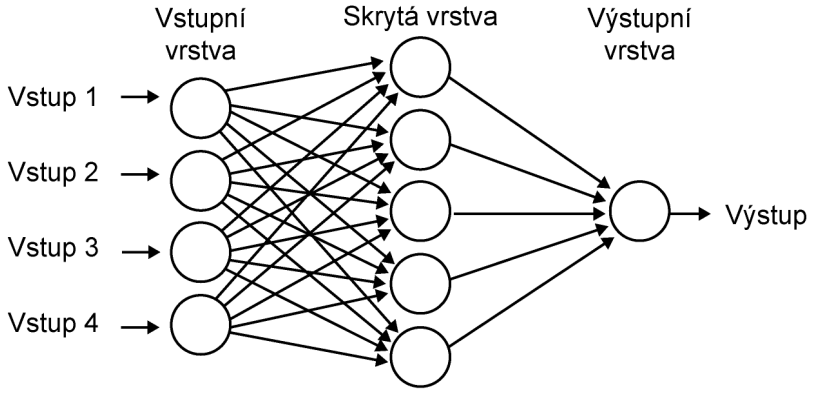
Obrázek 5: Zjednodušené zobrazení funkce Particle filter. Zdroj: [14]

výšku, ve které se letadlo právě pohybuje, a součet vzdálenosti od povrchu a nadmořskou výšku v daném bodě povrchu. Rozdíl těchto hodnot[38] dosadíme do distribuční funkce normálního rozdělení $N(\mu = 0, ?)$, ze které získáme pravděpodobnost s jakou se letadlo na daném místě nachází. Vypočtenou pravděpodobnost označíme za váhu. Tímto byla částice ohodnocena a tento postup opakujeme u všech částic, čímž vznikne ohodnocená populace. V dalším kroku algoritmu se váhy *normalizují*, to jednoduše znamená, že provedeme součet všech vah a váhy tímto součtem podělíme.

Nyní přichází druhá iterace, kde již částice nerozesejeme náhodně, nýbrž "nejslabší" jedince odstraníme a "silné" jedince posílíme (*resampling*). Tento krok provedeme tak, že vytvoříme novou prázdnou populaci, do které přidáváme částice výběrem z předchozí populace podle jejich váhy (tj. pravděpodobnosti). Tímto způsobem nám na jednom místě může vzniknout několik částic. V nové populaci mají všichni jedinci stejné fitness.

U každého jedince provedeme pohyb (*moving* či *prediction*) podle posledních naměřených hodnot ze senzorů. V některých případech je vhodné použít další statistické rozdělení, díky čemuž se jedinci nesdružuji na jednom místě, avšak rozmělní se po blízkém okolí od své původní

[38] $rozdil = nadmoraskaVyska_{letadlo} - (nadmoraskaVyska_{povrchu} + vzdalenostLetadlaOdPovrchu)$



Obrázek 6: Tří vrstvá neuronová síť

pozice. Například u letadla nepočítáme s tím, že letadlo může letět pozpátku, proto zařídíme, že nejvíce částic se bude posouvat v okolí směru letu.

Nyní znovu změříme každému jedinci váhu jako v kroku vyhodnocení.

Po druhé iteraci mohou být částice (resp. jejich hloučky) stále vzdálené, proto tento algoritmus opakujeme od resamplingu po normalizaci, kdy v ideálním případě postupně částice formují jeden hlouček.

4.3 Neuronové sítě

Umělé neuronové sítě (dále neuronové sítě, anglicky *Artificial neural networks*) jsou součástí posledního trendu obecně nazývaného jako strojového učení (anglicky *machine learning*). Ukázalo se, že neuronové sítě jsou schopny řešit problémy, jenž algoritmicky by byly velice složité. Koncept neuronových sítí však vznikl již v polovině minulého století[39] a v principu využívá obdobný koncept neuronových sítí jaký můžeme nelézt v biologickém mozku.

$$Y = S\left(\sum_{i=1}^{N}(W_i X_i) + \omega\right) \tag{5}$$

Neuronová síť je matematický model. Stavebním kamenem je *neuron*, který je popsán rovnicí 5, kde Y je výstupní hodnota neuronu, X_i je vstup neuronu, W_i je váha vstupu, S je přechodová funkce daného neuronu a ω je *bias* či práh daného neuronu.

4.3.1 Přenosová funkce

Neuron je definovaný přenosovou funkcí, která podle součtu vážených vstupů neuronu vrací výstup.

$$S(x) = \frac{1}{1 + e^{-kx}} \tag{6}$$

[39] `https://www.fi.muni.cz/usr/jkucera/pv109/2000/xneudert.html`

Obvyklým příkladem přechodové funkce bývá funkce *sigmod* popsaná podle vzorce 6.

4.3.2 Architektury

Mezi nejjednodušší neuronovou síť se řadí perceptron, který je složen z jednoho neuronu. Jeho omezením však je, že umí řešit pouze lineárně separovatelné problémy, jako je například logický součin či logický součet. V překladu to znamená, že klasifikuje vstupy pouze do dvou skupin rozdělených přímkou. Z tohoto důvodu vytváříme více neuronové a vícevrstvé sítě, které dokáží klasifikovat data do nelineárně oddělených skupin. Neuronová síť se skládá ze vstupní vrstvy, skrytých vrstev a výstupní vrstvy. Každá vrstva je poskládána z několika neuronů (viz obrázek 6), kde každý neuron z vrstvy n (první vrstva je vstupní vrstva) je výstupně propojený se všemi neurony vrstvy $n + 1$.

5 Mapové podklady

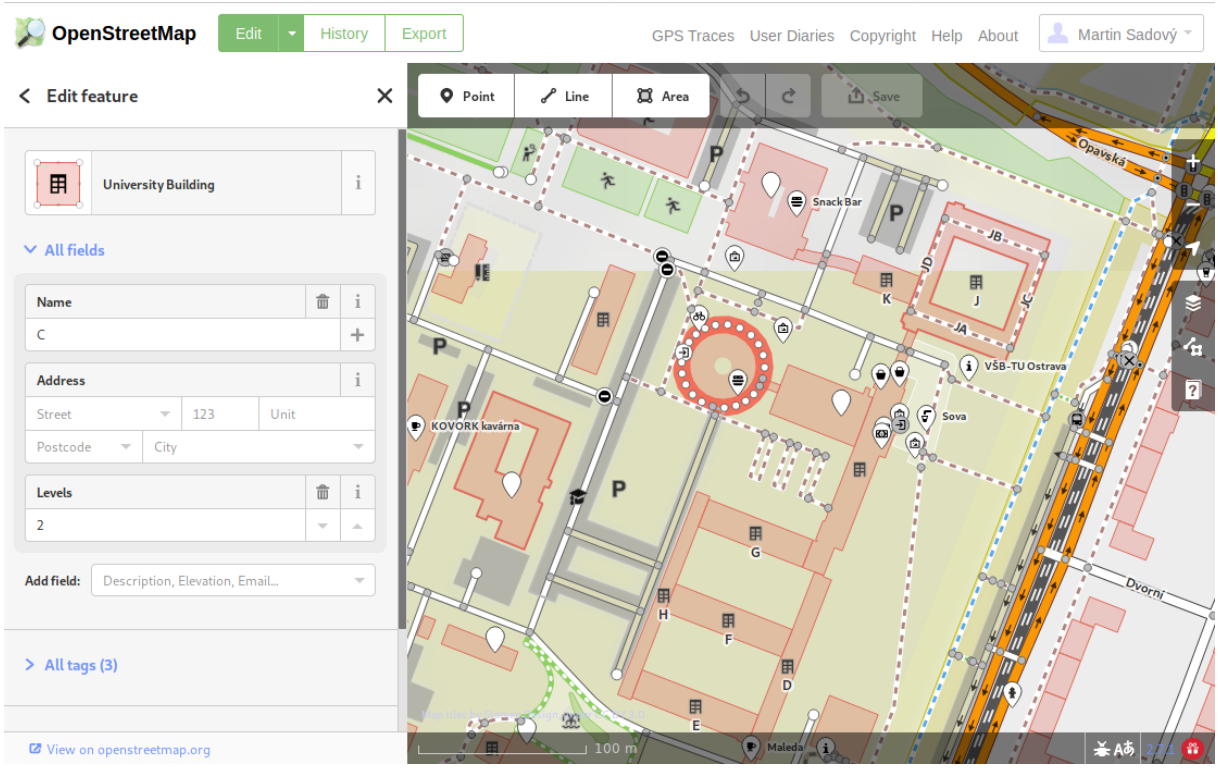
Obrázek 7: Ilustrativní ukázka tématické mapy, která ukazuje zábavnou formou známé oblasti na mapě České republiky. Snímek Mapy.cz k datu 1. 4. 2016.

Lidstvo používá mapy od nepaměti. Mapa obecně slouží k přehledu toho, kde se co nachází, a pro případnou orientaci. Mapy dělíme na místopisné (topografické), které popisují co nejpřesnější geografickou realitu. Ve větším měřítku tyto mapy označujeme jako všeobecně zeměpisné a nejsou tak přesné, protože informace ukryté v místopisných mapách generalizují. V případě, že chceme uživateli zobrazit informace podle určitého tématu - například poměr zaměstnanosti a nezaměstnanosti, hustotu zalidnění a jiné (viz obrázek 7) a nemusí se jednat o geografické rozdělení (tj. např. podle krajů) - používáme mapy tématické. Důležitým typem map jsou mapy katastrální. Ty označující přesné hranice pozemků (parcel) a dokumentují i věcná břemena.

Konkrétně katastrální mapy neposkytují detailní data o silnicích, pěších cestách, podnicích, parcích, atd. pro běžného uživatele. Z toho důvodu existují společnosti či skupiny, které se zabývají mapováním, ať už komerčně či zájmově. Jako příklad nejrozšířenějších komerčních map můžeme uvést Google Maps nebo, v případě České republiky, Mapy.cz od společnosti Seznam.cz. Mapy.cz používají jako zdroj informací taktéž nekomerční mapy. V tomto případě se jedná o OpenStreetMaps (OSM), které spravuje přes milion[40] dobrovolníků po celém světě a každý den se v těchto mapách provede tři miliony změn. V případě komerčních map máme možnost změnit obsah map, ale to obvykle pouze upozorněním na změnu, kterou následně provozovatel akceptuje a změnu do svých map zavede. Opačným případem je OSM, kde každý uživatel může změnit obsah map. Je však nutné mít na paměti, že základní pravidlo OSM říká, že přispěvatel nesmí použít data z komerčních zdrojů.

Pro editaci mapových podkladů v OSM existuje celá řada webových i nativních aplikací. Některé nástroje mohou podporovat jen základní editaci bodů (tzv. *Point of interest* (POI)), a to z důvodu, že mohou být součástí větší aplikace, jako je například navigace. Jiné nástroje

[40]Zdroj: `https://wiki.openstreetmap.org/wiki/Stats` k datu 13.3.2018



Obrázek 8: Editor map iD od OpenStreetMap

jsou zaměřené na editaci a mají plnou podporu standardizovaných vlastností. Mezi tyto aplikace se může řadit webový editor iD[41] (viz obrázek 8), nebo multiplatformní editor JOSM[42].

Podpora mapování vnitřních prostor však není nativní součástí ani jedné ze zmíněných aplikací, a to především z důvodu, že značky pro mapování vnitřních prostor se konsolidují v definicích a v minulosti existovalo několik návrhů, kterými šlo vnitřní prostory zmapovat. Naštěstí v obou případech existují projekty, které doplňují právě zmíněné aplikace o mapování vnitřních prostor. V případě editoru JOSM existuje zásuvný modul *indoorhelper*[43], který se až letos dočkal inovace, opravy chyb a aktualizace značek pro mapování vnitřních budov podle posledních definic z dokumentace OSM.

5.1 Úložné formáty a GeoJSON

Základní vlastností úložných formátů je geodetický systém. V následujících formátech nalezneme WGS84, který byl vydán ministerstvem obrany USA v roce 1984[44]. Zeměpisná délka (*longitude*) nabývá hodnot -180° do 180° (z východu na západ) a zeměpisná šířka (*latitude*) nabývá hodnot -90° do 90° (z jihu na sever).

[41] https://wiki.openstreetmap.org/wiki/ID
[42] https://wiki.openstreetmap.org/wiki/JOSM
[43] https://wiki.openstreetmap.org/wiki/JOSM/Plugins/indoorhelper
[44] https://en.wikipedia.org/wiki/World_Geodetic_System

OpenStreetMap poskytuje své mapové podklady ve vlastní struktuře ve formátu XML s příponou `.osm`. Při práci s geografickými daty se setkáváme s alternativním formátem GeoJSON, který najdeme pod RFC 7946[45]. Tento formát však není poskytován ze serverů OSM, avšak jeho výhodou je dobrá čitelnost a podpora díky tomu, že jeho struktura staví na formátu JSON. Formát GeoJSON ukládá souřadnice pouze ve formátu WGS84 z důvodu interoperability[46] a předpokladu, že zařízení nemá plnou databázi jiných geodetických systémů. Formát zápisu souřadnic je v pořadí zeměpisná délka (longitude), zeměpisná šířka (latitude) a volitelně nadmořská výška v metrech. Obvykle se zapisuje zeměpisná šířka jako první a na tuto vlastnost je třeba si dát při implementaci pozor.

Formát GeoJSON se skládá z `GeoJSON` objektu, ve kterém můžeme nalézt krajní hranice (`bbox`) a může nabývat typu `Feature`, nebo jejich kolekcí (`FeatureCollection`). `Feature` je popsán geometrickým tvarem typu bod (`Point`), řetězec úseček (`LineString`) či polygon (`Polygon`), a nebo jejich *multi* varianty, kde jeden objekt je popsán více body (`MultiPoint`), více řetězci úseček (`MultiLineString`) či více polygony (`MultiPolygon`). Volitelnou součástí, avšak pro tuto práci velice důležitou, je sekce *properties*, která má strukturu klíč-hodnota. Do této struktury zapisujeme všechny informace, které chceme o daném objektu mít při zpracovaní programem.

5.2 Mapování vnitřních prostor

Základem uživatelského rozhraní aplikace je zmapování vnitřních prostor, které nabízí uživateli výsledný požitek z navigace. Při zjednodušeném použití by jako mapový podklad postačil obrázek, který by byl na zobrazovací jednotku vložen jako vrstva. Tento postup by však postrádal jak možnost interakce uživatele s aplikací, tak to, že pokud máme mapové podklady vložené v dostatečné kvalitě, můžeme je využít v lokalizačních algoritmech a pro vyhledávání trasy.

Z tohoto důvodu bylo rozhodnuto, že je nutno překreslit podklady do mapových objektů, jako jsou například místnosti, chodby, výtahy, schodiště apod. Na trhu existují ve své podstatě dva majoritní provozovatelé mapových podkladů s podporou vnitřních prostor - komerční Google Maps a nekomerční a svobodný OSM. Pro účely této práce byl zvolen OSM s využitím *Simple Indoor Tagging*[47].

Mapování obvykle probíhá překreslením existující grafické dokumentace do výsledného elektronického mapového podkladu. Tuto dokumentaci lze získat z existujících papírových dokumentů, či u novějších staveb rovnou v elektronickém formátu.

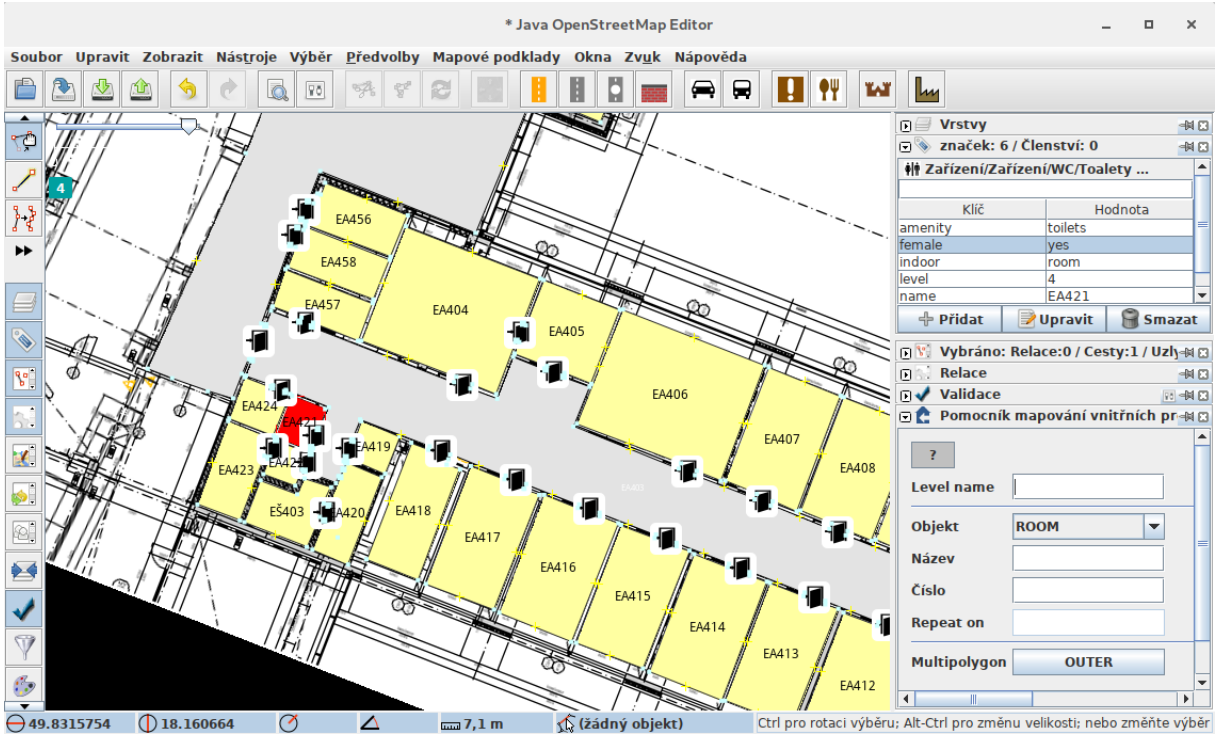
V této diplomové práci je k mapování využit editor JOSM, jenž je rozšiřitelný o užitečné doplňky, které zjednodušují a urychlují proces zakreslování. Grafické podklady v obrázkovém formátu se vloží do pomocné vrstvy, nad kterou se vyznačují vnitřní prostory. K tomuto účelu je vhodné využít doplněk PicLayer[48].

[45] `https://tools.ietf.org/html/rfc7946`

[46] V RFC se jde dočíst, že byl zvolen ještě jeden alternativní geodetický systém, ale ten nebyl z těchto důvodů ve výsledném dokumentu zahrnut.

[47] `https://wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging`

[48] `https://wiki.openstreetmap.org/wiki/JOSM/Plugins/PicLayer`



Obrázek 9: Příklad mapování uvnitř budov pomocí JOSM a indoorhelper, kde máme vybrané dámské toalety s podpůrnými mapovými podklady

V tomto doplňku je možné obrázek mapy přizpůsobit - rotací, změnou rozměrů či celkovou transformací pomocí společných bodů - přesným rozměrům v geografických materiálech (viz obrázek 9).

Výrazným rozdílem mezi klasickým mapováním venkovních prostor a vnitřních prostor je označování objektů do konkrétních podlaží. Z tohoto důvodu je nutné využít další doplněk, a to právě výše zmíněný *indoorhelper*. Tento doplněk zajišťuje, aby uživatel mapující prostory nepřicházel do kontaktu s objekty, jenž se nachází v jiném podlaží. Kromě této vlastnosti je rozšíření doplněno o nástroj usnadňující rychlé nastavení vlastností (*properties* zmíněné v sekci o GeoJson 5.1) v uživatelsky vybraném objektu, díky němuž není nutné složitě nastavovat vlastnosti typu *klíč-hodnota* u každého objektu, ale například vybereme objekt a označíme jej za místnost, nebo konkrétněji například za pánskou toaletu.

Avšak ne každá vlastnost je v tomto doplňku podporována, či dokonce je podporována odlišně od dokumentace v OSM. Z tohoto důvodu následující podkapitola objasňuje značky, které byly využity v této diplomové práci.

5.2.1 Základní značky v Simple Indoor Tagging

Základem vícepatrového mapování jsou podlaží. K tomuto účelu v Simple Indoor Tagging[49] slouží značka `level`[50]. Hodnota značky může nabývat číselné hodnoty podlaží a nebo rozsah podlaží. Jednoduchým příkladem je `level=3`, který udává, že objekt se nachází pouze ve třetím podlaží. Složitějším příkladem může být `level=-2--1;5;7`, což značí, že objekt spojuje podlaží (například výtah, jenž zastavuje pouze na:) -2, -1, 5 a 7.

Místnosti se obecně označují značkou `indoor=room`[51], avšak můžeme se setkat s hodnotou `indoor=corridor` u místnosti, jenž slouží primárně jako chodba. Tyto objekty jsou pouze ve formátu polygonu. Užitečnou vlastností je značka `name`, díky ní lze uživateli na mapovém podkladě zobrazit název místnosti, např. `EA409`. U místnosti a obecně POI lze vyznačit vybavenost objektu značkou `amenity`[52], díky čemuž lze místnost označit za toaletu `amenity=toilets` a rozlišit ji na pánskou `male=yes` nebo dámskou `female=yes` (aplikaci této značky na lze vidět na obrázku 9).

Kromě pěší cesty označené jako `highway=footway` existuje taktéž bodové označení pro výtah `highway=elevator`, který je povolen pouze v objektu typu bod či úsečky, nebo schody `highway=steps` (nikoliv schodiště jako prostor). Schody jsou postaveny z úseček, jenž utvářejí cestu. Tato cesta však nemusí být vytvořena směrem do vyššího patra, a proto existuje značka `incline`, jenž může nabývat `up`, nebo `down`. Schody se vkládají do místnosti typu schodiště, která představuje místnost se značkou `stairs=yes`.

Posledním základním objektem v mapování jsou dveře či průchod. Jedná se o bod se značkou `door`. U průchodu nabývá hodnoty `no`, případně může genericky nabývat `yes` či určitého typu[53]. Doplňkovou značkou na tomto bodě může být `entrance`, jenž může označovat hlavní vchod (`main`), únikový východ (`emergency`) a další[54].

[49] `https://wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging`
[50] `https://wiki.openstreetmap.org/wiki/Key:level`
[51] `https://wiki.openstreetmap.org/wiki/Key:indoor`
[52] `https://wiki.openstreetmap.org/wiki/Key:amenity`
[53] `https://wiki.openstreetmap.org/wiki/Key:door`
[54] `https://wiki.openstreetmap.org/wiki/Key:entrance`

6 Vývojové prostředí a nástroje

6.1 Rozdělení aplikace na moduly

Základní částí lokalizační služby je mobilní stanice, která v reálném čase zobrazuje polohu uživatele. Pro vývoj mobilní aplikace s vlastním lokalizačním algoritmem není ideální, aby aplikace byla jediným ladícím nástrojem algoritmů. A právě proto je vhodnější aplikaci a algoritmy oddělit do separátních modulů, díky čemuž lze algoritmy využít v nástroji pro ladění.

Mobilní aplikace je vytvářena pro operační systém Android. Platforma tohoto operačního systému využívá ve vývojářském prostředí jako výchozí automatizační nástroj *Gradle*[55]. Z tohoto důvodu byl zvolen i tento nástroj pro všechny moduly.

Lokalizační aplikace je rozdělena na následující moduly

- **Společná knihovna** obsahuje načítání mapových podkladů, lokalizační algoritmus a veškerý společný kód nezávislý na OS Android a uživatelském rozhraní.
- **Aplikace pro OS Android** využívá HW mobilního zařízení pro lokalizaci a vyhledávání trasy na mapě.
- **Ladící aplikace pro počítač** čte záznamy a zobrazuje veškeré informace z lokalizačního algoritmu.

Ke tvorbě aplikací byly významně použity nástroje jako je programovací jazyk Kotlin a nástroj pro reaktivní programování.

[55] `https://gradle.org/`

6.2 Kotlin

Kotlin je staticky typovaný programovací jazyk vyvíjený společností *JetBrains*. Název Kotlin vznikl podle názvu ostrova Kotlin, kde taktéž sídlí vývojářské středisko tohoto jazyka. Kotlin je primárně vyvíjen, aby fungoval v běhovém prostředí *Java Virtual Machine* (JVM) a byl plně interoperabilní s knihovnami vyvíjenými v jazyce Java. Avšak v případě využití čistě jazyku Kotlin, je možný překlad zdrojového kódu do jazyka Javascript, díky čemuž se stává Kotlin více než jen alternativou jazyku Java. Další inovací Kotlinu je kompilátor Kotlin/Native[56], jenž překládá zdrojový kód psaný v Kotlinu již do instrukční sady konkrétní platformy, ať už je to x86_64 (Linux, Mac a Windows), ARM (Linux, Android), či MIPS nebo dokonce WebAssembly (wasm32) jako zástupce instrukční sady pro běhové prostředí ve webovém prohlížeči mající za cíl přiblížit rychlost vykonávaného kódu k nativním instrukcím se stejnou ochranou počítače jako očekáváme u webových prohlížečů, neboli se kód vykonává v *sandboxu*.

Kotlin se stává jedním z velice populárních programovacích jazyků, což lze například vidět na *The PYPL PopularitY of Programming Language Index*[57], kdy od představení plné podpory v ekosystému Android na Google I/O 2017[58] se stal jazyk vyhledávaným a velice oblíbeným. Tomuto trendu napomáhá také to, že vývojář ve vývojovém prostředí Android Studio (či IntelliJ Idea) může svůj kód napsaný v jazyce Java jedním kliknutím převést do jazyku Kotlin, a tak rychle migrovat na moderní programovací jazyk, ze kterého se některé vlastnosti pomalu objevují i v posledních verzích jazyku Java.

Kotlin přináší nové koncepty[59] a napravuje problémy, jenž se z jazyku Java nejspíše nikdy neztratí. V první řadě, čeho si programátor může všimnout, že kód, který byl automaticky převeden z jazyku Java do Kotlinu, je výrazně kratší, což také může evokovat lepší přehlednost. Avšak počet konstrukcí, které v jazyce můžeme nalézt je nespočet. Tato diplomová práce si neklade za cíl představit veškeré možnosti jenž Kotlin nabízí, a proto v následujícím textu jsou zmíněné především vlastnosti, které byly v implementaci aplikace nejčastěji používány.

V roce 1965 při vzniku programovacího jazyku ALGOL W vznikl[60] ukazatel indikující, že hodnota neukazuje na validní objekt tzv. *null-pointer* (konkrétně v jazyce C je to 0). Tento přístup můžeme považovat za správný, avšak jak se ukázalo, tento postup vytváří mnoho neočekávaných chyb a dochází k následnému selhání programu. Kotlin se tomuto neduhu snaží jít čelem a rozhodl se do jazyku přidat speciální syntaxi (`?`), kde hodnota v programu může nabývat nulový ukazatel a postup jak se v takovém případě zachovat. Například v případě, že při volání metody či funkce dojde k tomu, že předaná hodnota nabývá *null*, avšak parametr funkce explicitně neudává, že se jedná o nulovou hodnotu, dojde k porušení integrovaného systému zajišťující *pre-condition* a tím pádem k selhání programu. Ač tato vlastnost zní opačně,

[56] `https://kotlinlang.org/docs/reference/native-overview.html`
[57] `http://pypl.github.io/PYPL.html`
[58] `https://youtu.be/Y2VF8tmLFHw?t=1h27m15s`
[59] `https://kotlinlang.org/docs/reference/idioms.html`
[60] `https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare`

než bychom očekávali, je to ochrana především kvůli interoperabilitě s Javou. Kotlin díky tomu přináší *null-safe* chování.

```
data class Customer(val name: String, val birthDay: Date)
fun Date.isToday(): Boolean {
    return DateUtils.isToday(this)
}
class Price(private val normalSale: Double) {
    lateinit var specialPrices: PriceStrategy
    fun priceCalculator(customer: Customer?): Double {
        return customer?.let {
            val price = when {
                customer.birthDay.isToday() -> specialPrices.priceFor(customer)
                else -> normalSale
            }
            price * 0.99
        } ?: 0.0
    }
}
```

Výpis 1: Názorná ukázka kódu v Kotlinu

Z funkcionálních jazyků si Kotlin vzal mnohé. Především je inspirovaný jazykem Scala. Hlavní dominantou jsou *high-order function* (HOF) a *lambda*. V mnoha jazycích se právě lambda (anonymní funkce) integrovala s velikou oblibou vývojářů. V případě kombinace HOF a rozšíření tříd o vlastní metody (*Extension method* známý především z jazyka C#) lze vytvářet *operace*, například z funkcionálního programování známá transformace `map` rozšiřující rozhraní `Iterable`. Mezi další metody přijímající lambdu patří `filter`, který filtruje prvky z kolekce, `groupBy`, jenž seskupuje prvky podle klíče z lambdy, ale taktéž metoda `reduce`, která kolekci spojí do jednoho výsledného objektu. Pokud bychom srovnali databázový SQL dotaz a takto poskládané funkce v Kotlinu, lze dojít k podobné expresivitě - délce kódu resp. dotazu.

Kotlin si taktéž odnesl z funkcionálního programování neměnnost hodnoty (*imutabilitu*). Například klíčovým slovem `val` vytváříme neměnnou "proměnnou"(opakem je `var`). Obdobně se definují vlastnosti (*properties*) tříd, a kolekce jsou rozděleny na *mutable* a *immutable*.

Obdobou struktur z jazyka C jsou datové třídy (`data class`), kterým se automaticky generují metody `hashCode()` a `toString()`.

Nespornou výhodou oproti jazyku Java je funkcionalita tzv. *smart-casting*. Kompilátor dedukuje datový typ proměnné bez potřeby explicitního přetypování. Tuto vlastnost nalezneme v řídících strukturách (`if`, `when`). Řídící struktury fungují rovněž jako funkce, které vrací hodnotu (viz kód 1 - řádek s konstruktem `when`).

Jako poslední vlastnost stojí za zmínku delegáti, jenž lze aplikovat na vlastnosti třídy. Příkladem může být delegát `lazy`, jenž inicializuje hodnotu vlastnosti až v čase, kdy kód programu hodnotu vyžaduje. Kotlin obecně podporuje *Domain-specific language* (DSL). Jinými slovy, programátoři si mohou v Kotlinu vytvořit co nejexpresivnější jazyk k dané problematice [61].

6.3 Reaktivní programování

Reaktivní programování (*ReactiveX*[62]) je založeno na událostech (*event-based program*) s asynchronním zpracováním s využitím návrhového vzoru pozorovatel (*Observable*). Můžeme si tedy představit zřetězené pozorovatele, kteří předávají a transformují data až na místo určení. Právě výhodou reaktivního programování je asynchronnost, což znamená, že program nečeká na data. Můžeme si to představit jako opak *Iterable*, kde program čeká než se data doručí (*pull*), zatímco *Obserable* je volán až data dorazí (*push*).

Výhodou konceptu je snadná skladba *Observables* nebo-li *operátorů* jako ve funkcionálním programování. V překladu to přináší, že je definovaný vstup a výstup toku. Data proudí ze vstupu volitelně transformované na výstup, díky čemuž data mohou být zpracovávána souběžně (od slova *concurrency*[63]). Vstupní i výstupní data nemusí být plynulá a mohou být doručována s velkými nepravidelnými časovými rozestupy.

Každý *Observable* obsahuje metodu `onNext`, která je volaná v případě další zprávy, ve které může dojít k odeslání zprávy dál (té samé nebo transformované).

Konečným uzlem toku je odběratel (*Subcriber*), jenž se zaregistruje k *Observable*, který chce odebírat. V principu se jedná o *Observable*, který dál již neemituje zprávy z toku a chová se jako výstup, který například zobrazí data uživateli.

Protože se jedná o koncept *push*, musí se určitým způsobem odběratel na výstupu dozvědět o chybě. K tomuto účelu existuje metoda `onError` a jedná se o závěrečné volání. Obdobným závěrečným voláním je i `onComplete`, jenž je doručeno odběrateli, jakmile již žádná data v daném toku nebudou emitována.

6.3.1 ReactiveX a Java - RxJava, a RxKotlin

Reaktivní programování je podporované v mnoha jazycích (resp. existuje v těchto jazycích oficiálně spravovaná knihovna) - například Java, JavaScript, .NET, Scala, Clojure či Swift[64]. V této diplomové práci je použita knihovna *RxKotlin*, která rozšiřuje *RxJava 2*[65] o prvky usnadňující reaktivní programování, aby výsledné volání vypadalo obdobně, jak je programátor v jazyce Kotlin zvyklý.

[61] `https://github.com/zsmb13/VillageDSL#a-weird-overkill-dsl`
[62] `http://reactivex.io/`
[63] `https://www.youtube.com/watch?v=cN_DpYBzKso`
[64] `https://github.com/ReactiveX`
[65] `https://github.com/ReactiveX/RxJava`

Každá implementace reaktivního programování může obsahovat další operátory, které ReactiveX nedefinuje, avšak v konkrétním jazyce jsou možné.

Autoři projektu RxJava se rozhodli druhou verzi plně přepracovat. Přibyl zde nový druh toku tzv. *Flowable*, který dokáže oznámit zdroji, že tok není plynule odbavován odběratelem, a aby zdroj zpomalil v emitování nových zpráv. Tento mechanismus se nazývá *backpressure*.

6.3.2 Zdroje

Zdroje toku lze vytvářet vlastní (např. jako roura pro odpověď na HTTP požadavek apod.), nebo lze využít již připravené, výčtem: `Range` emituje čísla od n to m, `Interval` emituje zprávy v časových intervalech a `Just` emituje set zpráv.

6.3.3 Operátory

Jak bylo zmíněno, operátory lze skládat, a protože ne všechny operace jsou přímočaré, jako ve funkcionálním programování, jenž implementuje Kotlin (`map`, `filter`, `last`, atd.), v této sekci budou objasněny odlišné a velice efektivní operace pro určité případy užití, jenž právě byly použity v této diplomové práci.

Velmi užitečným operátorem je `Buffer`, který si zapamatovává zprávy a ty následně emituje jako balíček. Buffer (nejen v RxJava) má více případů použití, buď emituje skupinu podle počtu (např. každých N vstupů emituje jako jeden výstup) zpráv nebo podle času. U času můžeme například chtít, aby každých N sekund emitoval nastřádané zprávy. Taktéž lze, aby emitoval každých N sekund, ale prvky pouze za posledních M sekund, kde $N < M$. Avšak může se stát, že bychom chtěli, abychom dostávali balíček zpráv co N sekund, jenž byla přijata v posledních K sekundách, kde $K > N$, i tuto vlastnost API dovoluje[66].

Některé operace (např. `CombineLatest`) mohou odebírat zprávy z více zdrojů, které následně transformují do jedné zprávy svým odběratelům. Příkladem může být ukázkový kompas[67], který vyžaduje naměřené hodnoty ze dvou senzorů, a následnou kombinací údajů z těchto senzorů určí orientaci.

Posledními dvěma zmíněnými operátory jsou `Sample` a `Throttle`, kteří zajišťují, že odběratel nebude zahlcen tokem zpráv, ale dostane poslední zprávu například každých 100 ms.

[66] Tuto vlastnost jsem dohledal až na neoficiálním internetovém blogu `http://www.introtorx.com/content/v1.0.10621.0/13_TimeShiftedSequences.html`

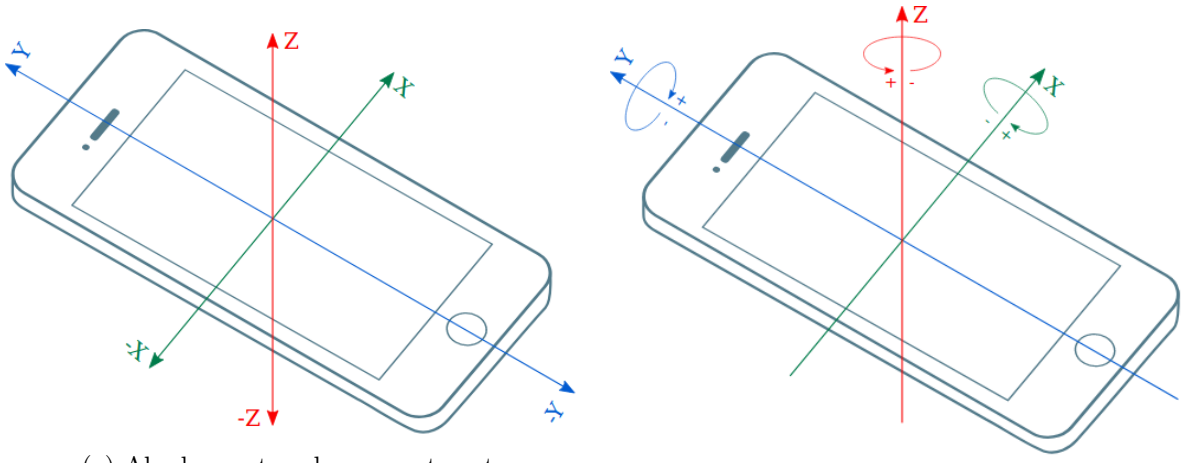
[67] `https://www.techrepublic.com/article/pro-tip-create-your-own-magnetic-compass-using-androids-internal-sensors/`

6.4 Hardwarové senzory u zařízení s OS Android

Mobilní aplikace využívající operační systém Android mají bohaté možnosti využití hardwaru na zařízení kde jsou provozovány. Společnost Google jako správce operačního systému udržuje dokument *Android Compatibility Definition Document* (zkráceně CDD[68]), ve kterém specifikuje potřebný hardware pro určitý typ zařízení včetně jeho parametrů.

Například Bluetooth a Bluetooth LE může být v zařízeních zahrnut. Právě z tohoto důvodu si lze při tvorbě aplikace nadefinovat, který hardware je používán a vyžadován. Operační systém při instalaci aplikace zkontroluje požadavky dané aplikace. V případě nevyhovění, dojde k odmítnutí aplikace a instalace skončí chybou.

6.4.1 Pohybové a poziční senzory



(a) Akcelerometr nebo magnetometr

(b) Gyroskop

Obrázek 10: Vyznačení významu os ze senzorů

Mezi pohybové senzory patří akcelerometr a gyroskop. K pozičním se řadí magnetometr.

Akcelerometr je senzor měřící zrychlení, které je udáváno v jednotkách m/s^2. Díky tomuto senzoru lze měřit chování uživatele a analyzovat a vyvozovat, které činnosti provádí (sezení, chůzi, běh, atd.). Z důvodu, že změny provedené a měřitelné senzorem se velmi rychle mění, je ideální hodnoty odečítat v krátkých intervalech.

Senzor akcelerometru `TYPE_ACCELEROMETER` je obvykle zatížen zemskou gravitační silou, z toho důvodu jsou i data (osy x, y a z viz obrázek 10a reprezentující směr pohybu) ve standardním senzoru pro akcelerometr zatížena silou země a to $g = 9.81 m/s^2$. Z tohoto důvodu, zařízení, které leží na stole, bude mít hodnotu na ose z zatíženou tímto číslem. Data bez tohoto zatížení lze získat ze senzoru `TYPE_LINEAR_ACCELERATION`, který tuto hodnotu odstraňuje, a to buď na základě výpočtu či gravitačního senzoru.

[68] `https://source.android.com/compatibility/cdd`

Gyroskop je senzor, jenž měří míru rotace zařízení v úhlové rychlosti rad/s. Hodnoty, které udává senzor v Android zařízení jsou v protisměru hodinových ručiček - viz obrázek 10b. Tento senzor lze například využít pro detekci otočení telefonu, a tím změny orientace obrazu.

Posledním zmíněným senzorem je **magnetometr**, jenž měří magnetickou indukci udávanou v μT. Jeho osy jsou obdobné jako u akcelerometru 10a.

Kombinací senzorů můžeme dosáhnout lepší přesnosti výsledných dat, nebo dat nových. Obecně se tomuto principu říká *Sensor fusion*. Příkladem může být vytvoření aplikace pro kompas, kde využijeme principu akcelerometru a magnetometru[69].

6.4.2 Odečítání senzorů v zařízení Android

Senzory lze odečítat různou rychlostí. U nastavení `SENSOR_DELAY_FASTEST` není četnost odečítání limitována a data jsou dodávaná aplikaci v co nejkratších intervalech (od 0 ms). Avšak minimální interval, který senzor zvládne, je udáván v metodě `getMinDelay`[70], a to v mikrosekundách. U nastavení `SENSOR_DELAY_GAME` jsou nové hodnoty odečítány od 20 ms intervalu. Obdobně je to u nastavení `SENSOR_DELAY_UI`, kde nové hodnoty jsou odečítány od 60 ms intervalu. Ve výchozím nastavení se používá `SENSOR_DELAY_NORMAL`, kdy hodnoty jsou odečítány od 200 ms intervalu.

Pro efektivnost aplikace je velice důležitá volba rychlosti senzoru, protože v případě častého odečítání, a tím pádem i častého volání kódu, který zpracovává data ze senzorů, může dojít k vysokému zatížení procesoru, nebo v opačném případě bez častého odečítání může dojít k nepřesnému vyhodnocení.

Pro odečítání hodnot ze senzoru se registruje instance třídy implementující rozhraní `SensorEventListener`. Toto rozhraní je generické a implementaci lze využít ke čtení hodnot z více senzorů. Hodnoty lze následně rozlišit typem senzoru, ze kterého data přišla.

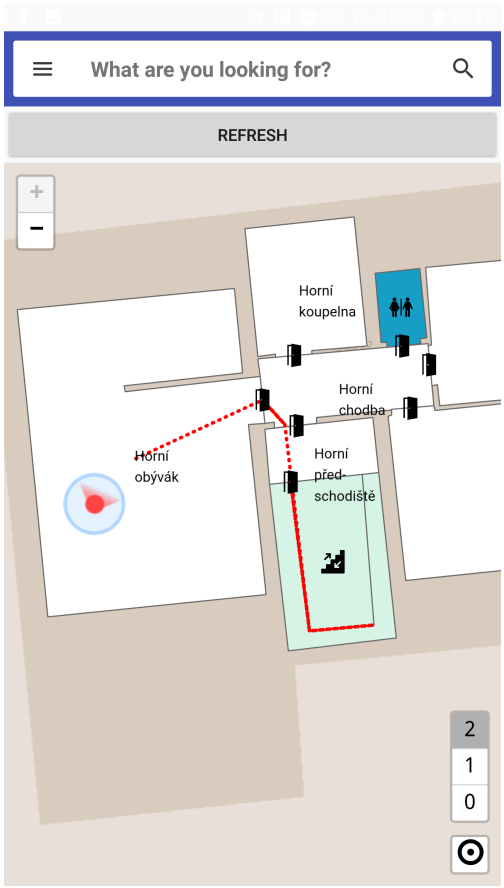
[69] S využitím dat poskytnutých ze senzorů a využití metod `getRotationMatrix` a `getOrientation` v `https://developer.android.com/reference/android/hardware/SensorManager.html`

[70] `https://developer.android.com/reference/android/hardware/Sensor.html`

7 Mobilní aplikace a její implementace

Mobilní aplikace slouží jako interaktivní nástroj pro uživatele k zobrazování mapových podkladů, zobrazení lokalizované pozice, a taktéž k vyhledání trasy k určitému cíli. V kapitole 2 jste byli seznámeni s aplikací, jež byla vyvinuta pro semestrální projekt. Navazující aplikace využívá z původní aplikace koncept zobrazovacího systému mapových podkladů. V první části je úvod do uživatelského rozhraní, po kterém následují technické detaily implementace.

7.1 Zobrazení mapy, pozice uživatele a trasy



Obrázek 11: Mobilní aplikace zobrazující polohu v obydlí s trasou k východu.

Uživatel je při otevření aplikace vyzván k výběru podkladu. Tento bod je však dobrovolný a slouží pouze v případě, že aplikace má více připravených lokalit - příkladem mohou být separátní podklady pro Ekonomickou fakultu v centru města Ostravy a pro kampus VŠB-TUO v Ostravě Porubě. Po následném výběru a načtení mapových podkladů se uživatel dostává do uživatelského rozhraní s mapou a vyhledávacím polem. Uživatel si může v rozhraní zvolit pod-

laží, přiblížit a oddálit mapové podklady. Uživatel může vyhledávat trasu ze svého místa na místo určení (východ, toaleta, učebna, atd.) a taktéž se dozví, kde a v jakém okruhu se nachází.

Dominantním prvkem v tomto uživatelském rozhraní jsou mapové podklady, které prošly vylepšením. Aplikace je tvořena s pomocí nástroje *Brunch*, který funguje jako automatizační nástroj pro vývoj webových aplikací. Zobrazení mapových podkladů zajišťuje aktualizovaná knihovna *leaflet* ve verzi 1.3.1. První viditelnou změnou (viz obrázek 11) je podpora zobrazování pozice včetně okruhu, ve kterém se uživatel může nacházet a orientací, na kterou stranu se uživatel nejspíš dívá. Tento úkon byl vytvořen za pomoci vlastní vrstvy, na kterou jsou naneseny další prvky - kruhy a značky.

Leaflet obsahuje dva typy kruhů, `CircleMarker`, jenž svůj poloměr definuje v pixelech, a druhý kruh je `Circle`, který svůj poloměr definuje v metrech. Kruh za pozicí je vytvořen pomocí `Circle` a pozice je vytvořena pomocí standardní značky (*Marker*) s vlastní ikonkou, jež znázorňuje orientaci uživatele. Avšak při výzkumu, jak tuto značku otočit, aby byla orientována s uživatelem, se zjistilo, že je vhodné použít rozšíření, protože leaflet rotaci nepodporuje [71] a kaskádové styly (CSS) ikonky jsou zapouzdřené a nelze je externě modifikovat.

Další novou vlastností je zobrazení cesty k cíli. Při první implementaci bylo použito rozšíření *Ant Path*[72] s efektní animací zobrazující směr, kterým se má uživatel vydat. Při průběžném testování se ukázalo, že toto rozšíření je nespolehlivé a způsobovalo značné komplikace v aplikaci. Z tohoto důvodu je uživateli zobrazena přerušovaná červená cesta (viz obrázek 11) pomocí `Polyline`[73], která je zakončena výrazným konečným bodem. V případě takového zobrazení je taktéž nutné dbát na podporu cesty napříč patry. Například v případě vícepatrového schodiště, které je potřeba sejít či vyjít, je tímto cesta zobrazena v každém patře zvlášť.

Každá zmíněná komponenta je vytvořená na vlastní vrstvě (`L.Layer`[74]), která je rozšířena o podporu zobrazení podle vybraného podlaží.

Webová aplikace komunikuje s nativní aplikací pomocí JavaScriptové třídy, která je virtuální reprezentací metody napsané v Android aplikaci. Principiálně jde o obdobu *Java Native Interface*. Opačným směrem neexistuje možnost vytvořit rozhraní a je nutné využít volání JavaScriptové funkce z adresního řádku (např. `javascript:appEvent(serializované data v json)`). Zprávy, které webové rozhraní implementuje jsou:

- `position` - předává pozici uživatele a přesnost
- `loadmap` - předává mapové podklady pro zobrazení
- `showpoint` - vytvoří bod na mapě s popiskem
- `showpath` - vytvoří zobrazení cesty z bodu do bodu

[71] `https://github.com/Leaflet/Leaflet/issues/143`
[72] `https://rubenspgcavalcante.github.io/leaflet-ant-path/`
[73] `http://leafletjs.com/reference-1.3.0.html#polyline`
[74] http://leafletjs.com/reference-1.3.0.html#layer

7.1.1 Zpracování senzorů

Nová aplikace začala zpracovávat data z více senzorů a z tohoto důvodu byl celý systém zpracování dat ze senzorů přepsán do reaktivního stylu, díky čemuž nebylo nutno vytvářet specifickou třídu implementující rozhraní `SensorEventListener`, ale nově jsou data ze senzorů poskytována jako tok dat.

```
val rxBluetooth = RxBleClient.create(this)
val bleReactive = rxBluetooth
        .observeStateChanges()
        .startWith(rxBluetooth.state)
        .switchMap({
            when (it) {
                RxBleClient.State.READY -> rxBluetooth
                        .scanBleDevices(scanSettings)
                        .onErrorResumeNext(Observable.empty())
                else -> Observable.empty()
            }
        })
        .subscribe {
            println("%s: %d".format(it.bleDevice.macAddress, it.rssi))
        }
```

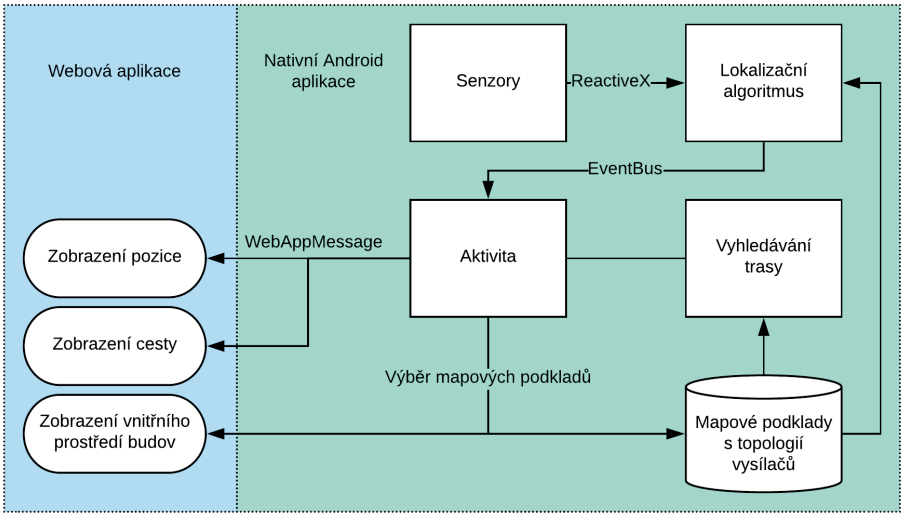
Výpis 2: Reaktivní zaregistrování skenneru okolních BLE zařízení

Výhodou pro programátora je, že nemusí ošetřovat kód aplikace podle stavu BLE senzoru. Příkladem je kód 2, ve kterém kód čeká na změnu stavu BLE adaptéru. V případě stavu `READY`, transformace ve funkci `switchMap` vrátí nový tok nalezených zařízení z BLE adaptéru. V opačném případě vrátí prázdný tok.

Specifický případ může nastat, když uživatel vypne Bluetooth adaptér z rychlého nastavení v notifikační liště a OS Android do původního toku vyšle chybu, tuto chybu kód zachytí a ukončí vstupní tok dat (původně vytvořený metodou `scanBleDevices`), avšak stále je zachován tok výstupní. V případě, že uživatel BLE adaptér opět zapne, je tok dat obnoven.

7.2 Architektura aplikace

Aplikace v předchozí verzi obsahovala službu, která znala všechny informace o topologii vysílacích stanic a obsahovala algoritmy pro výpočet polohy. Koncept služby je i v této verzi zachován s rozdílem, že problematiku lokalizace řeší společný modul obsahující veškeré algoritmy. Služba nyní funguje jako vrstva mezi lokalizačními algoritmy a uživatelským rozhraním se vstupními daty senzorů.



Obrázek 12: Architektura aplikace

Data ze senzorů se transformují na zprávy, které jsou předávány do lokalizační části přes `Flowable` toky (viz kód 2). V lokalizační části se data ze všech senzorů transformují. V případě dat z senzoru BLE se k naměřeným hodnotám připojí poloha vysílací stanice, nebo se hodnoty z neznámých zařízení vyfiltrují.

V časových intervalech je spuštěn lokalizační algoritmus, který si přečte naměřené hodnoty a provede lokalizaci. Následně vypočtenou polohu vrací přes `Flowable` tok zpět do služby.

Poloha je ze služby předána aktivitě přes rozhraní `EventBus`[75], které zajišťuje komunikaci mezi vlákny pro uživatelské rozhraní a vlákny pro procesy na pozadí.

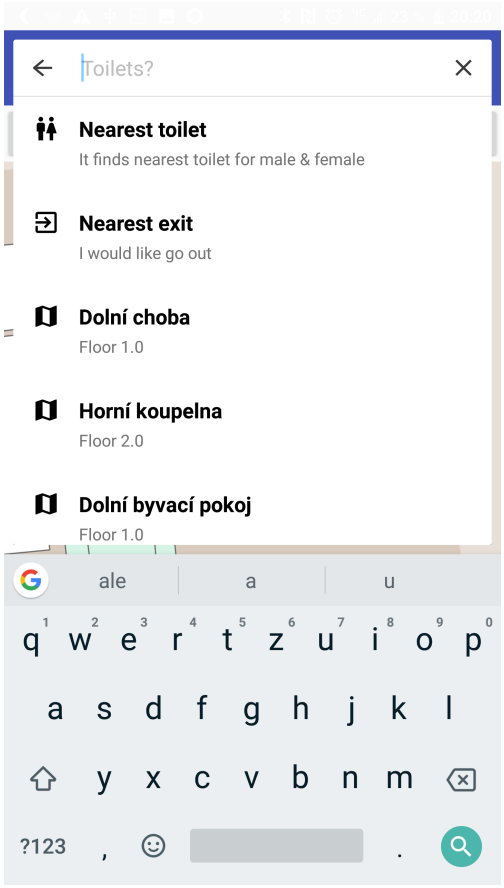
Aktivita přijatou polohu předá webové aplikaci pomocí zprávy (viz obrázek 12).

Aktivita se ke službě připojuje pomocí spojení `ServiceConnection`[76]. Služba přijme spojení pomocí metody `onBind`, ve které služba předá `LocationServiceBinder`[77], který slouží pro *request-response* komunikaci se službou. Jedná se o tzv. *Bound Service*[78]. To znamená, že služba je spuštěna při prvním spojení (`onBind`) a následně vypnuta při odpojení (`onUnbind`) posledního spojení.

7.3 Vyhledávání

Vyhledávací pole v Android aplikaci lze vytvořit za pomoci *AppCompact* (podpůrná knihovna vydávaná společně s Android SDK), která do horní lišty s menu dokáže přidat vyhledávací pole. Další možnou alternativou pro zobrazení vyhledávání je knihovna *MaterialSearchBar*, jež

[75] `https://github.com/greenrobot/EventBus`
[76] `https://developer.android.com/reference/android/content/ServiceConnection.html`
[77] `https://developer.android.com/reference/android/os/Binder.html`
[78] `https://developer.android.com/guide/components/bound-services.html`



Obrázek 13: Vyhledávání místa v mobilní aplikaci.

dokáže zobrazit seznam položek z vyhledávání přes celou obrazovku. Vzhled jednotlivých položek v seznamu je včetně nápovědy potřeba doimplementovat pomocí adaptéru `SuggestionsAdapter`.

Vyhledávání nabízí konkrétní objekty (např. místnosti či východy), nebo například nejbližší toaletu či východ (viz obr 13). Po výběru uživatelem vyhledaného místa, je uživateli navržena trasa z jeho pozice k vyhledanému místu.

Detailnějšímu popisu implementace algoritmu pro vyhledávání se věnuje následující kapitola 8.

8 Vyhledávání a interpretace mapových podkladů

Novým prvkem v uživatelském rozhraní je vyhledávání místa, ke kterému by měla být navrhnuta nejoptimálnější trasa. K dosažení této vlastnosti však nestačí pouhé vyhledání bodů, které jsou jednoduše položeny do obrázku, avšak je potřeba znát topologické rozložení budovy. Z tohoto důvodu si aplikace musí vytvořit vnitřní interpretaci mapových podkladů. Není to však jediný důvod, výhodou proč tento krok provádíme je podpora lokalizačního algoritmu, který tyto informace taktéž využívá. Převod z původních mapových podkladů do interní podoby je navržen v několika fázích, kdy v poslední fázi můžeme provádět vyhledávání trasy.

8.1 Převod do kartézského systému

Mapové podklady jsou dodávané v textovém formátu GeoJSON s OSM vlastnostmi (viz kapitola 5). GeoJSON formát však není vhodný pro vyhledávání, a proto tento formát převádíme do interní reprezentace. V GeoJSON se využívá souřadnicový systém WGS84, reprezentující zemský geoid v úhlech. Práce v úhlovém souřadnicovém systému není triviální záležitostí a bylo by potřeba mimoto přizpůsobit většinu algoritmů. Z tohoto důvodu je lepší najít mapovou projekci, kterou se dá souřadnicový systém v podkladech převést do kartézského systému, kde jedna jednotka představuje jeden metr. Problémem jakékoliv kartografické projekce je, že nereprezentuje reálný svět, a vždy porušuje nebo-li zkresluje [79] některé z následujících vlastností zobrazení:

- ekvidistantní - stejné vzdálenosti v určitém směru
- ekvivalentní - zachovává poměr ploch
- konformní - zachovává úhly

Pokud bychom hledali mapovou projekci pro vybraný prostor tak, aby splnila alespoň jednu ze zmíněných vlastností, je ideálním nástrojem *Projection Wizard*[80].

Při rozsáhlém průzkumu mnoha mapových projekcí se jako zajímavou možností projevila "mapová projekce" *Univerzální transverzální Mercatorův systém souřadnic* (UTM), která se skládá ze dvou souřadnic a čísla zóny. Souřadnice jsou v mřížce, která je obdobou kartézského systému a lze provést výpočet euklidovské vzdálenosti mezi dvěma body v metrech[81]. Nejedná se však o mapovou projekci. UTM rozděluje svět do zón, které se v určitých částech mohou překrývat, a z tohoto důvodu zmíněny výpočet vzdálenosti je platný pouze v případě, že oba body jsou ve stejné zóně.

V každé zóně je využito *Transverzální Mercatorovo zobrazení* s rozdílnými parametry. Právě této vlastnosti se využívá v převodníku. Převodník pro určitou mapu je definovaný středem. K vyhledávání středu se používá algoritmus K-Means, kde výsledný střed se vypočte jako průměr

[79] `https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=59996`
[80] `http://projectionwizard.org/`
[81] `https://web.viu.ca/corrin/FRST121/Help/UTM%20Coordinates.pdf`

mezi shluky podle jejich mohutnosti. Díky tomuto postupu se nestane, že menší odlehlý objekt bude mít výrazný vliv na výsledný střed.

8.2 Převod objektů do interní podoby

V dalším kroku je potřeba převést objekty zapsané v GeoJSON do podoby interních objektů. Interní objekt (obecně je dále nazvaný jako *Element*) reprezentuje například místnost, dveře, schody, chodbu, cestu, výtah atd. Všechny tyto elementy musí mít extrahované vlastnosti z původního GeoJSON.

Specifickým případem je *majáček* popsaný v kapitole 3. V této práci jsou majáčky převedeny do mapových podkladů, což usnadňuje jejich správu a organizaci v editoru JOSM. Majáček je definovaný bodem a popsán vlastnostmi:

- `leguide:beacon` definuje, že daný bod je majáček. Zatím podporuje pouze hodnotu `ble` reprezentující BLE majáček. Ta však v budoucnu může být rozšířena o další varianty resp. technologie, jako je například Wi-Fi.
- `leguide:beacon_ble_mac` identifikátor BLE majáčku.

Každý element by měl reprezentovat určitý geometrický tvar. Například schody jsou složené z úseček, místnosti jsou polygony nebo dveře jsou body. K reprezentaci těchto objektů se používá knihovna *The JTS Topology Suite*[82] (zkráceně JTS), která implementuje mnoho užitečných funkcí a operací s těmito geometrickými útvary. Příkladem takovéto operace může být spojení elementů z jednoho podlaží do jednoho tvaru, a to pomocí operace `union`, kde na výstupu dostaneme jeden geometrický objekt.

8.3 Vytváření grafů pro vyhledávací algoritmus

Důležitý předpokladem ve vyhledávacích algoritmech je mít data reprezentovaná ve struktuře, která je ideální formou pro algoritmické vyhledávání. Mapové podklady se však skládají ze spojitých prostorů - například místnost je reprezentovaná polygonem. Prohledávání ve spojitém prostoru by na mobilním zařízení mohlo vést k vysoké neefektivitě. Z tohoto důvodu je potřeba převést výše zmíněné elementy do grafu, který je reprezentován vrcholy a hranami.

Mapové podklady vždy byly zdrojem k vyhledávání tras. V mapách však nenalezneme konkrétní trasy z bodu do bodu, ale obecné cesty jako jsou automobilové silnice, vlakové tratě, či cesty pro pěší. Z těchto cest lze následně určitým algoritmem vytvořit optimální trasu. Při vyhledávání tras v široce využívaných aplikacích lze narazit na trasu, která však není optimální a lidskou zkušeností lze najít trasu lepší. Je to z toho důvodu, že například některé chodníky pro chodce nejsou zaneseny do mapových podkladů.

[82] `https://github.com/locationtech/jts`

Právě z tohoto důvodu v této aplikaci bylo potřeba vymyslet rozdílný systém pro vyhledávání trasy, který by si nezakládal na ručním zmapování cest, ale na algoritmickém prohledání prostoru pro vyhledávání tras.

Při zmapování prostoru však není výhodné vytvořit algoritmus, který by navrhoval nejkratší trasu. Takováto cesta by mohla vést k tomu, že uživatel by byl veden podél zdi, ač místnost nabízí mnohem více prostoru pro pohyb.

Naivní implementace by mohla například vypadat tak, že do prostoru se náhodně rozmístí body, u kterých si zkontrolujeme, zda jsou vzájemně viditelné. To by ovšem mohlo vést k tomu, že z některého místa by žádný bod nebyl viditelný. V překladu by to vedlo ke stejnému problému, jako když bychom prostor mapovali ručně.

Z tohoto důvodu bylo rozumné se podívat po algoritmu, který tento problém řeší efektivně. Po rozsáhlém průzkumu však žádná existující varianta neodpovídala potřebám. Proto byl vytvořen vlastní algoritmus.

8.3.1 Převod elementů do grafu

Při vytváření výsledného grafu je vhodné mít přehled, který element (dveře, místnost, chodba, atd.) je propojený s kterým elementem podle podlaží, například dveře a dvě místnosti (např. chodba a učebna). K tomuto účelu slouží graf `OverviewGraph` s vrcholy označující elementy.

Po přípravě zmíněného grafu je vybrán každý vrchol a v tomto vrcholu je vytvořen podgraf reprezentující cesty uvnitř vrcholu, a to podle druhu elementu.

V základu každý podgraf poskytuje dvě množiny vrcholu, hraniční (`connectible`) a vyhledatelný (`findable`). Tyto dvě množiny nejsou disjunktní, obvykle však hraniční vrcholy jsou podmnožina vyhledatelných vrcholů. Při vyhledávání trasy z bodu do bodu se námi zadané body připojí k vyhledatelným vrcholům. Zatímco připojitelné vrcholy slouží k propojení okolních elementů, tímto rozdělením dojde k tomu, že nemusíme u všech sousedících elementů provádět kontrolu viditelnosti všech vrcholů v podgrafech těchto dvou elementů.

Každý element však má své specifické tvorby podgrafů, tímto se následující řádky budou zabývat.

Dveře jsou bod reprezentovaný vrcholem. Tento vrchol je hraniční i vyhledatelný.

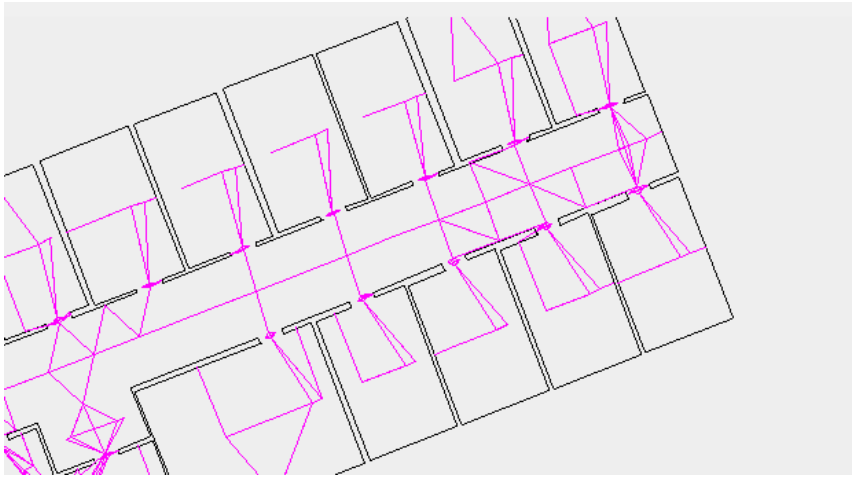
Cesta reprezentuje ručně vytvořenou cestu v mapových podkladech (užitečná například v případě propojení budov). Obě dvě množiny vrcholů jsou stejné. Mezi body v cestě je vážená hrana s hodnotou odpovídající vzdáleností bodů v metrech.

Schody jsou prvním elementem, u kterého se zmíněné množiny liší, protože se jedná o řetězec bodů ve dvou poschodích. V závislosti na tom, zda jde o schody stoupající, nebo klesající, je potřeba vybrat správné koncové body řetězce pro správné poschodí. Například schody byly zakresleny ve směru do nižšího patra a s nastavenou vlastností `incline=up`, díky které víme, že poslední bod patří do vyššího patra. Vážení hran je stejné jako u cesty [83].

[83] Zde si však uvědomuji, že chůze do schodů a ze schodů je odlišná, a taktéž ve své podstatě stejně vzdálená jako v případě chůze po rovném povrchu.

Výtah propojuje více podlaží, zastávky výtahu jsou jak hraniční tak vyhledatelné. Z důvodu, že výtah nezastavuje vždy v každém podlaží, každý vrchol má hranu s každým vrcholem v tomto podgrafu. Hrana je ohodnocená aproximovanou hodnotou, která vychází z výšky podlaží (například 2.5 m), rychlosti osobních výtahů (do jednoho metru za sekundu) a připočtení zpoždění nastávající při zpomalení a zrychlení.

Nejdůležitějším elementem jsou **místnosti a chodby**. Místnosti obecně mohou být zatočené a mohou obsahovat zdi, což může vytvářet bludiště. Z tohoto důvodu bylo zapotřebí vytvořit podgraf, kde vrcholy podgrafu jsou z jakékoliv části místnosti dostupné a jejich hrany tvoří cestu ven, díky čemuž lze uživateli nabídnout cestu z bodu do bodu na kterémkoliv místě mapy.



Obrázek 14: Ukázka výsledné mapy v budově FEI, kde každá místnost je propojena přes dveře s chodbou. Všechny tyto hrany byly vygenerovány pomocí Conforming Delaunay Triangulation

Ke tvorbě hran v místnosti by například mohla vést naivní implementace vytvořením mřížky v polygonu. Avšak při tvorbě mřížky se vytváří pro každý vrchol 8 hran, které je nutné zkontrolovat, zda nepřekrývají zdi. První implementace využívala metodu `cross` v JTS. Ukázalo se, že tato metoda nedokáže zajistit kvalitní kontrolu, zda úsečka mezi body není přerušená zdí [84]. Z tohoto důvodu byla použita funkce `intersection`, kde v případě křížení polygonu a úsečky vznikne odlišný objekt, než je původní úsečka. Bohužel tato metoda byla natolik výpočetně náročná, že bylo potřeba najít jiný algoritmus.

K tomuto účelu je použit algoritmus *Conforming Delaunay Triangulation*, který prostor polygonu (místnosti) rozdělí na trojúhelníky a to tak, že maximalizuje minimální úhly všech trojúhelníků v triangulaci [85]. Střed každé strany trojúhelníku přetváříme na vrchol s výjimkou krajních stran [86], který se propojí (vytvoří hranu v podgrafu) s ostatními takto vytvořenými vrcholy v daném trojúhelníků. V případě, že strana je součástí i dalšího vytvořeného trojúhel-

[84] `https://gis.stackexchange.com/questions/108498/check-if-line-crosses-a-polygon`
[85] `https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk5.pdf`
[86] Strana je částí strany triangulovaném polygonu

níku, pak je i tento vrchol stejný, z čehož plyne, že existuje cesta z jakékoliv strany do jakékoliv strany takto společných trojúhelníku. Tyto hrany jsou následně vážené jako v případě cesty.

Výsledný obraz hran lze vidět na obrázku 14.

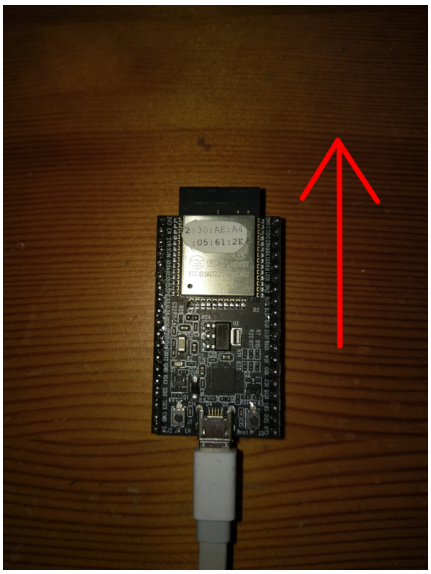
8.4 Vyhledávací algoritmus

Ve výsledném grafu lze vyhledat trasu za pomoci uspořádaného prohledávání (například A* nebo Dijkstrův algoritmus). V tomto případě je zvolen Dijkstrův algoritmus a rychlost nalezení trasy je na mobilním zařízení okamžitá.

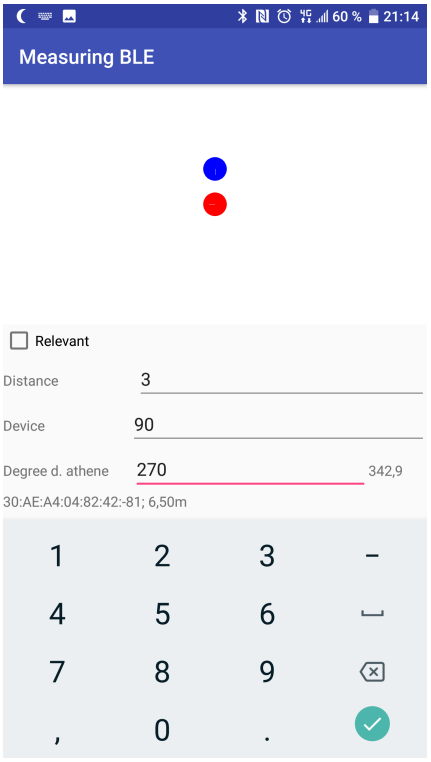
9 Vzdálenost mobilního zařízení podle charakteristiky signálu

Každé vysílací zařízení má svou vysílací charakteristiku signálu, z toho důvodu nelze pro lokalizaci mobilní stanice využívat generický vzorec, který převádí sílu přijatého signálu na vzdálenost v metrech. Proto bylo zapotřebí provést měření signálu a vytvoření vlastní exponenciální křivky reprezentující vztah naměřené hodnoty RSSI a vzdálenosti mobilního zařízení od vysílací stanice.

9.1 Měřící software



(a) ESP32 modul šipka směřující na sever



(b) Měřící software vytvořený pro zaznamenávání BLE signálu. Červený bod je vysílací stanice, modrý mobilní stanice. Oba body obsahují bílou čáru znázorňující úhly v parametrech. Červený obsahuje šipku doleva; Modrý dolů.

Obrázek 15: ESP32 vývojový kit, který je nasměrován na sever a taktéž bílá šipka v červený bod v pravém obrázku je nasměrován na sever

Vysílací stanice jsou postavené na platformě ESP32 a mobilní stanice na platformě Android. Pro měření bylo zapotřebí vyvinout vlastní aplikaci (vyobrazená na obrázku 15b), která bude přijímat MAC adresu vysílací stanice a společně se sílou signálu zaznamenávat do datového formátu k pozdějšímu zpracování. Aplikace je pracovně nazvaná `Measuring BLE`.

Aplikace zaznamenává do textového formátů CSV následující údaje:

- časové razítko v milisekundách

vzdálenost (m)	medián (dBm)	průměr (dBm)	směrodatná odchylka (dBm)
0.1	-46.0	-45.48	6.35
0.5	-52.0	-54.03	6.88
1.0	-62.0	-62.32	5.97
2.0	-65.0	-64.84	3.30
3.0	-66.0	-66.58	3.69
4.0	-72.0	-72.85	4.56
5.0	-79.0	-78.29	5.03
6.0	-81.0	-79.95	4.58
7.0	-79.0	-78.17	4.49
8.0	-79.0	-78.08	5.53
9.0	-82.5	-81.46	4.90
10.0	-79.0	-78.35	5.19

Tabulka 2: Statistická analýza naměřených hodnot z prvního měření

- zda je měření relevantní a nejde o momentální přesun uživatele mezi body
- MAC adresu vysílací stanice
- sílu signálu (RSSI), jenž byla přijata
- vzdálenost od vysílače měřená metrem v metrech
- úhel v jakém zařízení stojí oproti vysílací anténě
- v jakém úhlu je mobilní zařízení nasměrováno
- údaje z kompasu v mobilním zařízení (pro případné zpracování či vizualizaci v budoucnu)

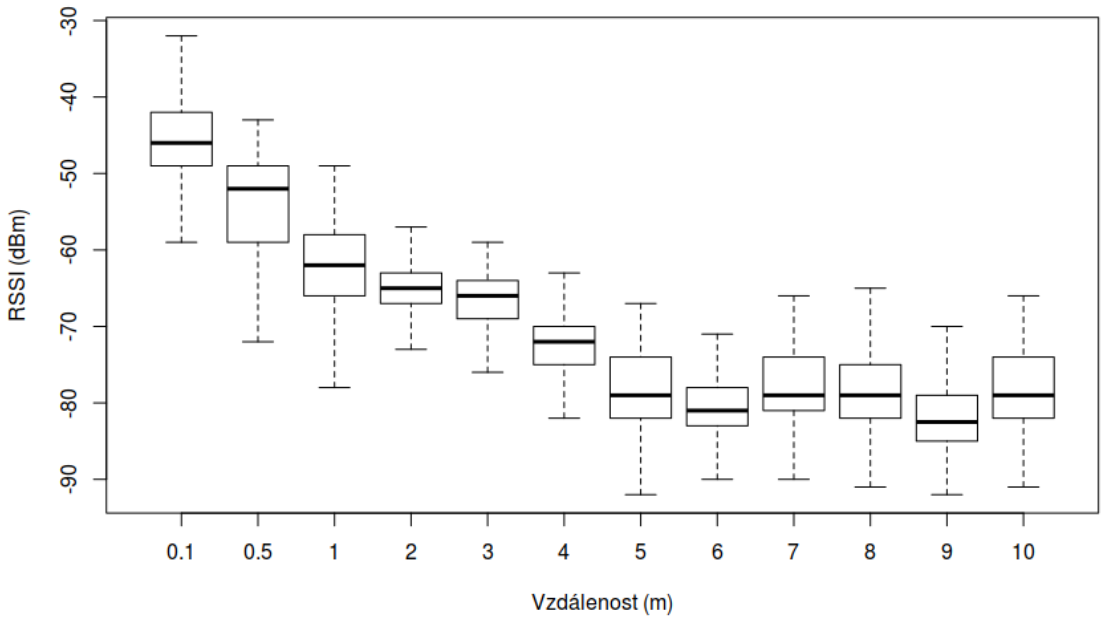
Každý záznam je vytvořen z události z BLE senzoru a jedná se o naměřená data, která neprošla žádným filtrem.

Pro zjednodušení analýzy jsem zvolil orientaci antény na sever (nultý stupeň) (viz obrázek 15a) a taktéž zaznamenaný údaj o poloze zařízení a orientace zařízení se odvíjí od zmíněného stupně nula - viz obrázek 15.

9.2 Měření

Měření bylo provedeno ve venkovním i vnitřním prostředí. Venkovní prostředí bylo zvoleno z důvodu, že obsahuje minimum odrazových ploch. Taktéž bylo ve venkovním prostředí provedeno mnohem více měření z více stran, natočení a vzdáleností. K měření bylo použito zařízení HTC 10.

Zaznamenaná data bylo nutno vložit do analytického programu, ve kterém lze vizualizovat data podle vzdáleností, či úhlu uživatele. Pro tento případ k základní vizualizaci postačí například RStudio (jazyk R), do kterého bylo nutno data nahrát, vyfiltrovat a následně vizualizovat. K vizualizaci byl zvolen krabicový graf (*boxplot*).

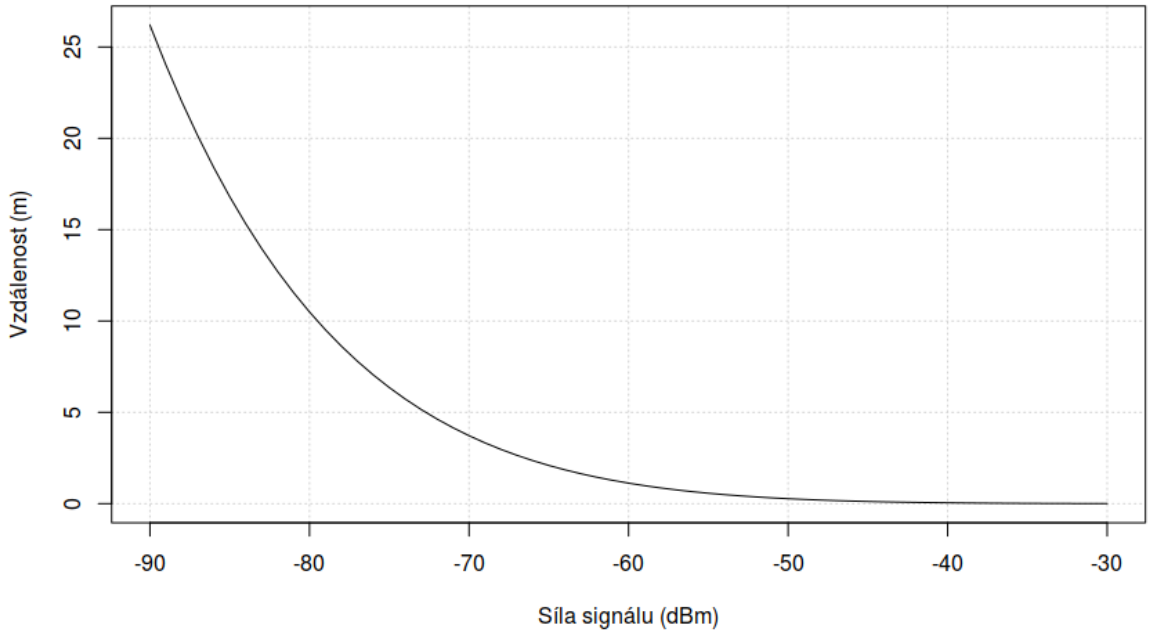


Obrázek 16: První měření. Naměřené hodnoty zanesené do krabicového grafu, kdy mobilní stanice je orientována ve směru na jih a vysílací stanice na sever, tedy bez útlumu lidské osoby. Odlehlá pozorování byla odstraněna.

Z grafu 16 vyplývá, že vzdálenost podle RSSI je rozlišitelná do 5 metrů. Směrodatnou odchylku naměřených hodnot naleznete v tabulce 2.

Ostatní grafy jsou přiložené v příloze A (graf 22 a 23) znázorňující naměřená data v ostatních polohách. Další měření probíhalo ve vnitřních prostorách a výsledek vypadal obdobně jako v předchozím zmíněném měření viz příloha A graf 24.

9.3 Analýza dat a vytvoření křivky



Obrázek 17: Vypočtená křivka podle naměřených hodnot.

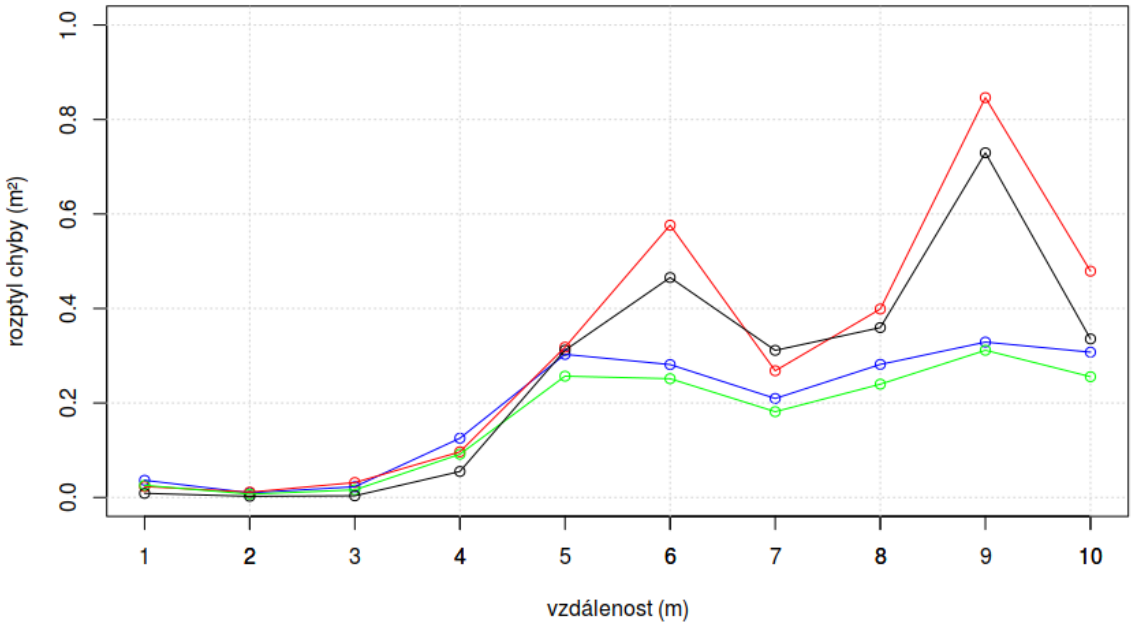
$$f(x) = 1.747292 \times 10^{-14} \times (-x)^{7.765577} \tag{7}$$

Z analýzy byly vybrány mediány naměřených hodnot a vloženy do nástroje [87], jenž dopočítá koeficienty a a b pro křivku $y = ax^b$ podle vstupních bodů. Vznikla křivka zobrazena v grafu 17 reprezentována funkcí f v rovnici 7.

V pozdějším testování se ukázalo, že zmíněná křivka nejlépe reprezentuje vzdálenost i v případě, že uživatel je orientován i opačným směrem. Z tohoto důvodu se stala křivka prioritní a další křivky nejsou ve výsledném algoritmu využity.

[87] `http://mycurvefit.com`

9.4 Zlepšení výsledků pomocí filtrů



Obrázek 18: Testování výsledné křivky s průměrovým (modrá), mediánovým (červená), Kalmanovým filtrem (zelená), Kalmanovým filtrem s mediánovou hodnotou za úsek (černé)

Naměřená síla signálu je kolísavá i při stacionární pozici mobilního zařízení. Může docházet k pohybu a otáčení uživatele s mobilním zařízením a tím pádem i možným útlumům přes tělo uživatele.

Během měření docházelo taktéž k občasnému přerušovanému snímání z BLE zařízení, kdy aplikace z OS Android nedostávala např. na půl sekundy žádné informace (někdy i déle), i když vysílače (z celkových 6 vysílacích stanic) byly aktivní. Toto chování může být způsobeno ovladači daného zařízení, nebo optimalizačními technikami v operačním systému.

Z těchto důvodů je vhodné použít filtr, který chybějící hodnoty aproximuje a zároveň neposkytne během krátkého intervalu informaci o vzdálenosti s velkým rozptylem.

Prvním jmenovaným filtrem, který tento problém řeší může být průměrový filtr, který si pamatuje naměřené hodnoty v čase zpětně, a ty následně využije k řešení zmíněných problémů. Možnou výhodou ale i nevýhodou tohoto filtru je zapomínání nejstarší hodnoty. Díky tomu lze například aproximovat vzdalující se mobilní stanici od majáčku. Obdobným filtrem je mediánový filtr se stejným principem zapamatovávání hodnot.

Posledním jmenovaným filtrem je Kalmanův filtr (KF). Tento filtr má nespornou výhodu v předpovědi budoucí hodnoty, a to i v případě, že poslední naměřené hodnoty obsahují výrazný rozptyl.

Další možnou optimalizací v mobilním prostředím je kombinace Kalmanova filtru a mediánového filtru (KMF), kdy zásobník ukládá hodnoty například za posledních 100 ms. Z těchto hodnot vypočte medián nebo průměr, které následně poskytne Kalmanovu filtru ve fázi korekce.

Zmíněné filtry byly otestovány. KF byl nastaven s koeficienty $Q = [0.001]$ a $R = [3.0]$. A KMF s koeficienty $Q = [0.1]$ a $R = [3.0]$.

KMF vykazoval nejlepší odhad do 4 metrové vzdálenosti (viz graf 18). Při reálném testování výsledného algoritmu Kalmanův filtr i KMF vycházeli jako nejlepší možné řešení.

10 Návrh lokalizačního algoritmu

V první části této kapitoly jsou popsány možnosti využití filtrů (z kapitoly 4) pro lokalizaci. Následující část se zabývá vytvořením lokalizačního algoritmu s aplikací zmíněných filtrů.

10.1 Možnosti využití filtrů pro zpřesnění výsledků

10.1.1 Kalmanův filtr

Kalmanův filtr (KF) lze použít pro lineární modely. Prvním příkladem je filtrování naměřených dat, které mohou být kolísavé z důvodu ohybu, odrazu a rozptylu naměřeného signálu. [6]

10.1.2 Particle filter

Particle filter (PF) oproti Kalmanovu filtru dokáže řešit i nelineární modely. PF se využívá v robotice, kde robot je vložen do neznámého prostředí a postupně z naměřených hodnot ze senzorů zjistí svou polohu. Tento koncept využití lze obdobně aplikovat i pro vnitřní prostředí budov, kdy mobilní aplikace dostává zkreslené údaje ze senzorů a nemá možnost využít globálních satelitních systémů. [8] Výhodou PF je, že z naměřených hodnot dokáže vytvořit více možných pozic. Postupným měřením nových hodnot a postupnou iterací se méně pravděpodobné pozice vyřazují.

10.1.3 Neuronové sítě

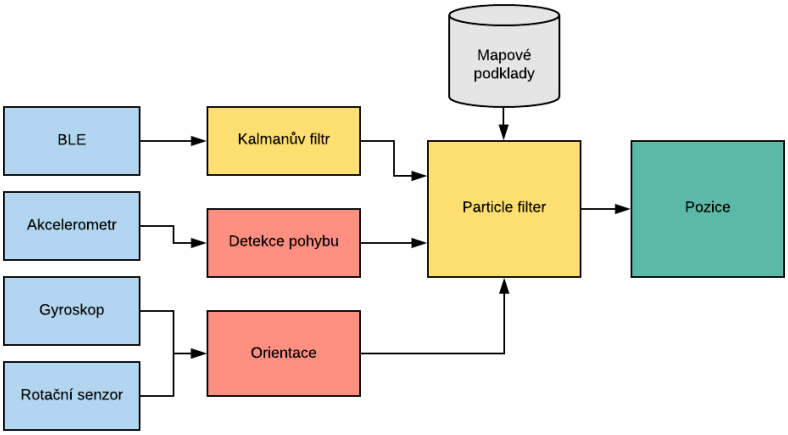
Neuronové sítě (NN) mají nespočet využití díky jejich přizpůsobivosti. Možným příkladem využití neuronové sítě může být naučení NN podle sekvencí náměrných RSSI a vzdálenosti, čímž může dojít k lepší detekci vzdálenosti [4] [5] [6]. Komplexnějším řešením je možnost rozdělit mapové podklady na menší segmenty a v těchto segmentech udělat měření hodnot v mřížce (tzv. *fingerprint*). Tímto lze dosáhnout vyšší přesnosti, avšak tato metoda vyžaduje vysokou náročnost na zpracování - sběr "fingerprintů" a následné učení NN pro každý prostor. [9] [10]

Kromě výpočtů vzdáleností od majáčku lze neuronovou síť využít k detekci rychlosti pohybu člověka, tento postup však vyžaduje mnoho trénovacích dat. [7]

10.2 Návrh algoritmu pro lokalizaci uživatele

Při zhodnocení využití lokalizace byl zvolen pro tvorbu výsledného lokalizačního algoritmu Particle filter. Při tvorbě algoritmu bylo potřeba dbát na fakt, že zdroje informací jsou nepřesné a mohou být silně zarušené.

Particle filter využívá teorii pravděpodobnosti pro ohodnocení částic (viz kapitola 4.2). Pro ohodnocení částic lze zvolit vlastní pravděpodobnostní rozdělení. Jedno rozdělení se stejnými parametry však pro tento případ nestačí a je potřeba kombinovat více pravděpodobnostních



Obrázek 19: Blokový diagram výsledného algoritmu

rozdělení s rozdílnými parametry. V tomto případě normální rozdělení definujeme jako $N(\mu, \sigma)$, kde μ je střední hodnota a σ je směrodatná odchylka.

V prvním kroku algoritmus rovnoměrně sampluje po prostoru částice, kde prostor je omezený polygonem celého podlaží.

Následuje krok *pohybu* v PF. V tomto kroku je vybrána každá částice a vytvořena nová částice v okolí vybrané částice podle normálního rozdělení. Pokud se uživatel nepohybuje je normální rozdělení definováno jako $N(0, \frac{1}{3})$, jinak $p \sim N(0, \frac{\omega}{3})$, kde rychlost chůze ω je nastavena na 0.6 m/s a následně podělena třemi, a to z vlastnosti normálního rozdělení, kdy v $\mu \pm 3\sigma$ by se mělo vyskytovat 99.7% všech vygenerovaných hodnot. Směrodatná odchylka je nastavena právě z důvodu, aby rychlost pohybu odpovídala rychlosti člověka a nedocházelo k příliš rychlému pohybu při ztrátě signálu. V tomto kroku byla taktéž při pohybu využita informace o orientaci, avšak tento přístup se později ukázal jako mylný. Důvodem byla rychlá konvergence polohy daným směrem.

Vážení každé vygenerované částice probíhá podle několika koeficientů.

Koeficient podle vzdálenosti od vysílacích stanic (z_b):

$$z_b = \sqrt{\sum_{i=1}^{n} F_b^i(-|d_i|)^2} \tag{8}$$

, kde n je počet poskytnutých vzdáleností od majáčků v danou chvíli poskytnutých z Kalmanova filtru, který byl vytvořen v kapitole 9.4, d_i je vzdálenost od majáčku i v metrech a $F_b^i(X)$ je distribuční funkce pro normální rozdělení $N(0, \frac{d_i}{3})$. V této formuli nalezneme princip trilaterace s robustnějšími vlastnostmi. Naměřená hodnota z vysílací stanice může být zkreslená zesílením nebo útlumem, a proto může docházet k tomu, že trilaterace výslednou pozici špatně vypočte. Výhodou PF je, že váha částice je vyhodnocena na její pozici, a proto "většina" blízkých kružnic,

reprezentujících vzdálenost od vysílače, dokáže společně více "přitahovat částice". Díky čemuž nedochází ke skokům, a to z důvodu, že částice plynule přecházejí do nejbližších nejpravděpodobnějších míst.

Koeficient podle pohybového vektoru (z_m). V případě, že v čase není detekován pohyb je koeficient nulový.

$$z_m = \sqrt{F_c(-|r|)^2 + F_a(-|\alpha|)^2} \tag{9}$$

, kde F_c je distribuční funkce normálního rozdělení $N(0, \Delta t \times \frac{\omega}{3})$, r je vzdálenost dané částice od poslední vypočtené pozice. F_a je distribuční funkce normálního rozdělení $N(0, 45)$ a α je vzdálenost dvou úhlů, a to, úhel ve stupních dané částice od poslední vypočtené pozice a úhel orientace. Δt je rozdíl času od poslední iterace. ω je rychlost chůze za sekundu.

Podpůrný koeficient standardní trilaterace (z_l):

$$z_l = F_l(-|l|) \tag{10}$$

, kde F_l je distribuční funkce normálního rozdělení $N(0, 2)$, l je euklidovská vzdálenost mezi částicí a optimálním bodem z trilaterace. z_l je nulový pokud optimální bod je vně jakékoliv místnosti nebo nelze trilateraci provést (např. nedostatek naměřených vzdáleností od majáčku). Výhodou tohoto koeficientu je možné zpřesnění při prvotních iteracích, taktéž může dojít k vychylování částic směrem, kterým by se ubírala klasická trilaterace.

Existence koeficientu podle lokace.

$$z_e = \begin{cases} 1.5 & \text{pokud částice je ve stejné místnosti jako z předchozí pozice} \\ 1.2 & \text{pokud částice se nachází v prostoru chodby} \\ 0.0 & \text{pokud částice je mimo mapu, nebo ve zdi} \\ 0.8 & \text{jinak} \end{cases} \tag{11}$$

[88] Koeficient z_e má mnoho významů. V prvním případě zabraňuje případnému jednoduchém průchodu přes zeď, avšak nemá bránit průchodu do jiné místnosti pokud uživatel prochází přes průchozí místa. Dalším důvodem proč tento koeficient zavádíme je, že uživatelé převážně chodí v chodbě a v případě, že by například vznikly dvě stejně pravděpodobné množiny částic podle kruhů (např. sekretariát a chodba) dojde k lokalizaci uživatele na chodbě.

Výsledná váha částice (w_i) je následující:

$$w_i = z_e * (z_l + z_b + \frac{z_m}{|d| + 1}) \tag{12}$$

, kde $|d|$ je velikost množiny vzdáleností.

[88] První hodnota od shora je vybranou hodnotou. Hodnoty vychází z ladění.

10.2.1 Implementace

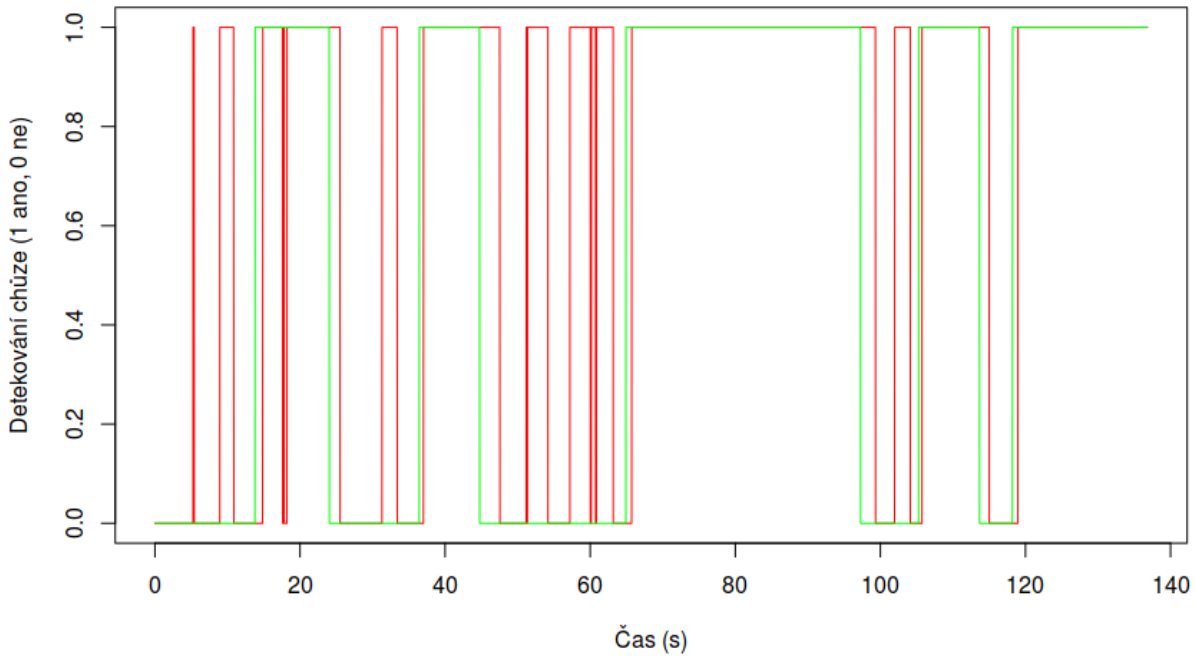
Implementaci lze nalézt ve společném modulu pod třídou `ReactiveProcess`. Tato třída přijímá Flowable toky a poskytuje jeden výstupní tok `positionReactive` s lokalizovanou polohou (viz obrázek 19).

Ve třídě lze nalézt metodu `reactiveBeaconsDistance`, která vytváří Flowable transformaci naměřených hodnot do vzorků (`Sample`). Metoda využívá třídu[89], která vytvoří konkrétní transformaci fungující jako filtr (mediánový, průměrový nebo v kódu zvolený Kalmanův - viz kapitola 9.4), do které jsou vzorky emitovány.

Ve třídě nalezneme metodu `reactiveBeaconsParticleFilter`, která obdobně vytváří transformaci s voláním Particle filter využívající model `ParticleFilterBuildingProbability`. Tento model je popsán výše v navrženém algoritmu. Model dále zajišťuje znovu rozesetí částic, pokud se pozice delší dobu výrazně odchyluje od bezstavové trilaterace.

Toky zdrojů (vzdálenosti majáčků, orientace, detekce pohybu a detekce podlaží - viz obrázek 19) se následně spojují v jeden. Dále PF zpracovává každých 100 ms[90] tento tok a emituje lokalizovanou polohu jako centroid všech částic se směrodatnou odchylkou vzdáleností všech částic od centroidu reprezentující přesnost.

10.3 Zdroj detekce pohybu



Obrázek 20: Detekce pohybu - zelená značí reálnou chůzi, červená detekovaný pohyb.

[89] Návrhový vzor stavitel.
[90] Tuto hodnotu lze změnit, testoval jsem taktéž jednu sekundu

Lokalizaci zařízení můžeme provádět na pohybujícím se nebo stacionárním objektu. V případě statického, může docházet ke zpřesňovaní polohy, zatímco u pohybujícího se objektu se může poloha aktivně měnit, a to znamená, že musíme brát v potaz větší chybu lokalizace a tím pádem i větší okruh, kde se sledované zařízení může nacházet. Z tohoto důvodu je vhodné využít senzory na detekci pohybu.

OS Android obsahuje senzor pro detekci kroků. Při testování senzor nedetekoval kroky okamžitě, ale informaci předával až po delší době (od půl minuty a více), což není slučitelné s lokalizací v reálném čase. Z tohoto důvodu bylo zapotřebí vytvořit algoritmus, který detekuje pohyb uživatele. První vytvořená aplikace zobrazovala křivku dat z akcelerometru pro analýzu.

K rozpoznávání kroků uživatele bychom mohli využít akcelerometru a postupným hledáním extrémů v křivce docházet k výpočtu kroků. Avšak pro algoritmus postačí detekce pohybu uživatele.

K tomuto účelu lze využít statistický rozptyl. Při následném průchodu a otestování této detekce se ukázalo, že detekce lze použít na základní rozlišení a při standardní chůzi dosáhla šestinovou chybovost, která však byla snížena díky následné filtraci. Detekce je pozitivní, jestliže směrodatná odchylka z naměřených hodnot z akcelerometru na osách x a y se za určitý interval pohybuje v rozmezí od 0.3 do 1.3.

10.4 Zdroj orientace

Kompas ve vnitřním prostředí budov vždy nedetekuje správnou orientaci uživatele. Pro zpřesnění otáčení orientace je využit gyroskop. Gyroskop udává změnu orientace v rad/s, avšak práce s orientací v prostoru nelze snadno reprezentovat Eulerovými uhly. V takovém případě by docházelo k chybné interpretaci [91]. Z tohoto důvodu se informace převádí do kvaternionů, což je 4 dimenzionální hyperprostor definovaný jedním reálným číslem a třemi imaginárními. K tomuto účelu je využita implementace [2] od Alexandra Pacha kombinující rotační senzor a gyroskop v Android zařízení.

10.5 Využití neuronové sítě

Možným způsobem určování polohy bylo využití neuronové sítě.

Prostor o velikosti 7x5 metrů byl rozdělen do čtvercových segmentů o velikosti 1 m^2. V každém segmentu byly naměřeny hodnoty z okolních třech majáčků.

Architektura neuronové sítě byla postavena ze čtyř vrstev. Vstupní vrstva přijímá 10 normalizovaných hodnot síly signálu pro každý majáček (dohromady 30 neuronů). Každá skrytá vrstva obsahuje asi dvě třetiny neuronů oproti předchozí vrstvě (20 a 10). Vstupní a skrytá vrstva má aktivační funkci *sigmod*. Výstupní vrstva je postavena ze dvou neuronů pro normalizované souřadnice x a y s lineární aktivační funkcí *identity* [92].

[91] `https://www.youtube.com/watch?v=C7JQ7Rpwn2k`

[92] Tato síť byla konzultována s Ing. Davidem Ježkem Ph.D.

Data byla rozdělena na dvě trénovací množiny:

- První množina obsahovala normalizovaná uspořádaná naměřená data.
- Druhá množina obsahovala normalizované hodnoty vytvořené normálního rozdělení, kde μ je medián naměřených hodnot, σ je směrodatná odchylky naměřených hodnot.

V obou případech trénovací množina obsahovala stejný počet vzorků pro každý segment.

K učení a tvorbě neuronové sítě byla využita knihovna *deeplearning4j* (ve verzi 0.8[93]. K učení byl využit univerzitní server Merlin[94].

Po jednodenním učení (1934100 iterací) s první trénovací množinou, se síť naučila s MSE[95] chybou 0.008303. Tato chyba představuje vzdálenost až 7 m. U druhé trénovací množiny síť vykazovala obdobné výsledky.

Po konzultaci a zhodnocení byla tato metoda vyhodnocena jako neúčinná a neefektivní. Důvodem je nutnost pro každých N nejbližších vysílačů vytvořit neuronovou síť, přičemž chyba zmíněné naučené sítě je vyšší než v případě kombinace naměření vzdálenosti a použití Particle a Kalmanova filtru.

[93] Bohužel instalace knihovny do prostředí Android ve vyšší verzi nefunguje
[94] `http://wiki.cs.vsb.cz/index.php/Merlin`
[95] Mean squared error

11 Testování a vyhodnocení navrženého algoritmu

Algoritmus je vhodné odzkoušet v mnoha prostředích, ať jde o prostředí, kde uživatel prochází pouze v jednopodlažním prostředí nebo prochází přes více podlaží, či uživatel se dostane mimo signál.

Testování aplikace probíhalo v mnoha iteracích[96], kdy jednotlivé komponenty v algoritmu byly dolaďovány či byly zkoušeny nové možnosti.

11.1 Testování v reálném prostředí

Prvním testovacím prostředím byl rodinný dům. V algoritmu je nastaveno, že vzdálenosti od majáčku jsou akceptovány do pěti metrů (viz závěr z kapitoly 9.2). Testování probíhalo uvnitř prostoru (7 m a 5 m) se třemi majáčky na krajních bodech místnosti. Ukázalo se, že přesnost lokalizace mobilního zařízení se ve většině případů pohybuje do jednoho metru. V tuto chvíli lokalizační algoritmus data z gyroskopu a akcelerometru výrazně nevyužívá.

Dalším testovacím případem bylo prostředí při pohybu mezi třemi místnostmi. Majáček byl umístěn v jedné místnosti, která je propojena chodbou o délce 5 metrů a v další místnosti jsou další dva majáčky. Test začínal v prostředí s jedním majáčkem, kde se lokalizace pohybovala přibližně po kružnici protínající místnost s majáčkem. Uživatel postupně přešel do chodby, kde mobilní zařízení již nemá signál z majáčku a začnou se využívat senzory pro orientaci a pohyb. Uživatel postupně přichází do druhé místnosti, kde je rovněž lokalizován. Následný návrat do místa počátečního bodu nebyl pro algoritmus problém.

Testování ve více patrovém prostředí. Toto testování probíhalo v místnosti s majáčkem, kde uživatel opustil místnost na chodbu a zabočil na schodiště, kde již ztratil signál. V tento moment algoritmus začal zpracovávat data ze senzorů pro orientaci a pohyb z mobilního zařízení. Uživatel došel do mezipatra, kdy v polovině schodiště se lokalizace zastavila. Došlo k přepnutí mapy na nižší podlaží a uživatel i lokalizovaná pozice dále pokračovala k nejbližšímu majáčku, který byl umístěn na dolním patře schodiště.

Druhým testovacím prostředím je chodba katedry informatiky ve čtvrtém podlaží budovy fakulty FEI. Prostředí je osazeno sedmi majáčky.

Tento test započal u vchodu EA457 a EA424, následně došlo k chůzi k místnosti EA406, kde se uživatel i výstupní pozice zastavila. Poté uživatel pokračoval k místnosti EA409, kde se s mobilním zařízením zastavil. Otočil se a odešel k místnosti EA406, kde se opět porozhlédl, a vrátil se na počáteční místo. Algoritmus je otestován ve dvou případech, kdy lokalizační algoritmus dostává hodnoty vzdálenosti pouze do pěti metrů a bez omezení. V případě vzdáleností od majáčku bez omezení algoritmus úspěšně sledoval uživatele až na konec chodby a následně zpět k počátečnímu bodu.

[96] Několika měsíční testování.

Obrázek 21: Zelený bod přestavuje trilateraci s optimalizací nejmenších čtverců. Žlutý je vytvořený algoritmus postavený na PF.

11.2 Vyhodnocení

Síla signálu naměřená z BLE vysílačů je velice kolísavá, avšak díky Kalmanovu filtru je možné vzdálenost predikovat efektivně. Particle filter se chová velice obstojně při průchodu uživatele především pokud využíváme orientační a pohybové senzory, díky nimž může dojít ke korekci směru vydaných částic. Při testováni v prostředí třech majáčků s mobilním zařízením byl objekt lokalizován ve většině případů do jednoho metru.

Trilaterace s optimalizací nejmenších čtverců [97] vykazovala problematické chování v prostředí, kdy průsečíky kružnic vypočtených podle vzdáleností vycházely mimo budovu (viz obrázek 21). Navržený algoritmus zachovával pozici, případně pozice pokračovala ve směru orientace uživatele, dokud nedošlo k novému zafixování polohy uživatele podle lokalizace s pomocí majáčků. Navržený algoritmus podával v některých případech lepší výsledky než zmíněná trilaterace, avšak horší výsledky mohly přijít při špatném zafixování pozice.

[97] `https://github.com/lemmingapex/Trilateration`

12 Doporučení

Semestrální práce se zabývala lokalizací, kdy algoritmus využíval generický vzorec pro převod RSSI na vzdálenost v metrech. Tento přístup byl v této práci přehodnocen a byla vytvořena specifická křivka podle naměřených hodnot. Pro úplnost je vhodné zmínit, že ne všechna mobilní zařízení mají stejnou charakteristiku síly signálu podle vzdálenosti. Z tohoto důvodu je vhodnější v komplexnějších aplikacích sbírat data a vytvářet převodní vztah pro jednotlivé modely mobilních zařízení.

Při práci se ukázalo, že pro menší prostory je vhodnější větší naměřené vzdálenosti nepropagovat do lokalizačního algoritmu. Tyto hodnoty mohou být zatížené velkým útlumem, avšak mobilní zařízení jej stále dokáže změřit a vytvořit tak kružnici obepínající celou budovu.

U větších prostor mohou mít majáčky omezenou vzdálenost větší (po případném zvážení žádnou).

Google ve vývojové verzi Android P (9) představil podporu techniky *Round-time trip* pro technologii Wi-Fi, díky čemuž se může přesnost lokalizace pohybovat v rámci jednoho až dvou metrů[98]. Tohoto poznatku by mohly budoucí práce využívat.

[98] `https://android-developers.googleblog.com/2018/03/previewing-android-p.html`

13 Závěr

Po přečtení této práce by měl čtenář být schopen vytvořit Android aplikaci a majáčky pro lokalizaci uživatele v budově pomocí technologie Bluetooth Low Energy. V práci se čtenář taktéž dozvěděl jakým způsobem vytvořit znovupoužitelné mapové podklady, které se dají využít pro účely lokalizačního algoritmu a vyhledávání trasy k určitému bodu v mobilní aplikaci.

Ve výzkumu bylo ukázáno, že síla signálu uvnitř budov je velice kolísavá a je zatížena chybou. V práci byl využit Kalmanův filtr, díky němuž naměřená výsledná vzdálenost od vysílače byla přesnější a méně citlivější na výkyvy naměřených hodnot.

V případě využití Particle filtr se po dlouhém procesu ladění podařilo, že určování výsledné pozice uživatele je přesnější a výsledná pozice je méně citlivější na chybu způsobenou sílou signálu z majáčku, kde signál je zesílen nebo utlumen. Chování navrženého algoritmu se pro uživatele chovalo přívětivěji než v případě využitých algoritmů v semestrální práci.

Práce aktivně využívala moderní techniky pro zpracování dat, známé z funkcionálních jazyků, díky čemuž práce z mého pohledu byla více než zajímavá. Rozšířila mi obzory nejen v problematice lokalizace uvnitř budov, ale také jsem se podrobněji dozvěděl o možnostech využití filtrů. Práci hodnotím jako jednu velkou zajímavou a užitečnou zkušenost.

Literatura

[1] GALLAGHER, N. a G. WISE.: *A theoretical analysis of the properties of median filters.*. IEEE Transactions on Acoustics, Speech, and Signal Processing [online]. 1981, 29(6), 1136-1141 [cit. 2018-04-06]. DOI: 10.1109/TASSP.1981.1163708.
ISSN 0096-3518. Dostupné z: <http://ieeexplore.ieee.org/document/1163708/>

[2] A. PACHA: *Sensor fusion for robust outdoor Augmented Reality tracking on mobile devices* [online].
University of Augsburg - Institut für Software & Systems Engineering, 2013 [cit. 2018-04-06]. Dostupné z: <https://alexanderpacha.files.wordpress.com/2017/05/masterthesis-pacha.pdf> Master Thesis.

[3] K. ZÜLSDORFF *Where Am I? - Triangulation, Kalmanfilter and Monte - Carlo localization* [online].
Dostupné z: <https://tams.informatik.uni-hamburg.de/lehre/2017ss/seminar/ir/doc/Kim_Zuelsdorff_Localization_and_Path_Planning.pdf>

[4] YANG, Bo, Yuanyuan LU, Jiao WANG, Yongqiang ZHANG a Yuechao MA. *An improved RBF neural network algorithm to mitigate the distance error based on RSSI.* In: 2017 36th Chinese Control Conference (CCC) [online]. IEEE, 2017, 2017, s. 3759-3764 [cit. 2018-04-09]. DOI: 10.23919/ChiCC.2017.8027945. ISBN 978-988-15639-3-4.
Dostupné z: <http://ieeexplore.ieee.org/document/8027945/>

[5] ZHANG, Huiqing a Xiaowei SHI. *A new indoor location technology using back propagation neural network to fit the RSSI-d curve.* In: Proceedings of the 10th World Congress on Intelligent Control and Automation [online]. IEEE, 2012, 2012, s. 80-83 [cit. 2018-04-11]. DOI: 10.1109/WCICA.2012.6357843. ISBN 978-1-4673-1398-8.
Dostupné z: <http://ieeexplore.ieee.org/document/6357843/>

[6] XUHUI, Zhang, Gao BAOJIANG, Liu YUKUN, Wang JUAN a Chang HUIMIN. *The Optimization of RSSI-Neural Network Positioning Algorithm.* In: 2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control [online]. IEEE, 2014, 2014, s. 633-637 [cit. 2018-04-11]. DOI: 10.1109/IMCCC.2014.135. ISBN 978-1-4799-6575-5.
Dostupné z: <http://ieeexplore.ieee.org/document/6995105/>

[7] SONG, Yoonseon, Seungchul SHIN, Seunghwan KIM, Doheon LEE a Kwang H. LEE. *Speed Estimation From a Tri-axial Accelerometer Using Neural Networks.* In: 2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society [online]. IEEE, 2007, 2007, s. 3224-3227 [cit. 2018-04-11]. DOI: 10.1109/IEMBS.2007.4353016. ISBN

978-1-4244-0787-3. ISSN 1557-170X.
Dostupné z: <http://ieeexplore.ieee.org/document/4353016/>

[8] PELTOLA, Pekka, Chris HILL a Terry MOORE. *Particle filter for context sensitive indoor pedestrian navigation.* In: 2016 International Conference on Localization and GNSS (ICL-GNSS) [online]. IEEE, 2016, 2016, s. 1-6 [cit. 2018-04-11]. DOI: 10.1109/ICL-GNSS.2016.7533865. ISBN 978-1-5090-1757-7.
Dostupné z: <http://ieeexplore.ieee.org/document/7533865/>

[9] DANIŞ, F. a Ali CEMGIL. *Model-Based Localization and Tracking Using Bluetooth Low-Energy Beacons. Sensors* [online]. 2017, 17(11), 2484- [cit. 2018-04-12]. DOI: 10.3390/s17112484. ISSN 1424-8220.
Dostupné z: <http://www.mdpi.com/1424-8220/17/11/2484>

[10] Kuan-Wu Su, He-Yen Hsieh, Jen-Chieh Hsu, Bo-Han Chen, Che-Jui Chang, Jenq-Shiou Leu *Implementing an iBeacon Indoor Positioning System using Ensemble Learning Algorithm*
Dostupné z: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/Abstracts-Implementing-an-iBeacon-Indoor-Positioning-System-using_Ensemble-Learning-Algorithm.pdf>

[11] Mark HUGHES *Troubleshooting Tools for Your Next Bluetooth LE Project: Ubertooth and the Nordic nRF Sniffer. All About Circuits - Electrical Engineering & Electronics Community* [online]. [cit. 13.04.2018].
Dostupné z: <https://www.allaboutcircuits.com/projects/troubleshooting-tools-bluetooth-LE-project-ubertooth-Nordic-nRF-sniffer/>

[12] *BLUETOOTH SPECIFICATION Version 4.0* [online]. [cit. 13.04.2018].
Dostupné z: <https://www.libelium.com/forum/libelium_files/bt4_core_spec_adv_data_reference.pdf>

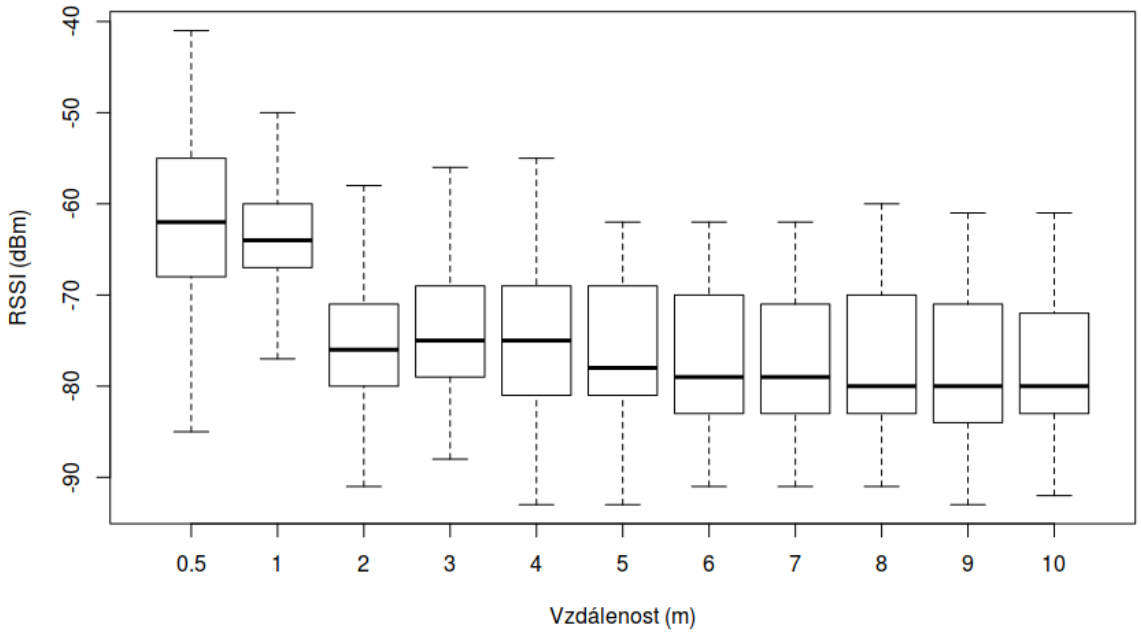
[13] D. HERNANDEZ, FERNÁNDEZ-CARAMÉS T., FRAGA-LAMAS P. a Escudero, C.J. *Design and Practical Evaluation of a Family of Lightweight Protocols for Heterogeneous Sensing through BLE Beacons in IoT Telemetry Applications* [online]. Copyright © 2008 [cit. 13.04.2018]. Dostupné z: <https://www.researchgate.net/publication/322092785_Design_and_Practical_Evaluation_of_a_Family_of_Lightweight_Protocols_for_Heterogeneous_Sensing_through_BLE_Beacons_in_IoT_Telemetry_Applications>

[14] MAGGIO, E. a A. CAVALLARO. *Hybrid Particle Filter and Mean Shift tracker with adaptive transition model.* In: Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005 [online]. IEEE, 2005, s. 221-224 [cit. 2018-04-13]. DOI: 10.1109/ICASSP.2005.1415381. ISBN 0-7803-8874-7.
Dostupné z: <http://ieeexplore.ieee.org/document/1415381/>

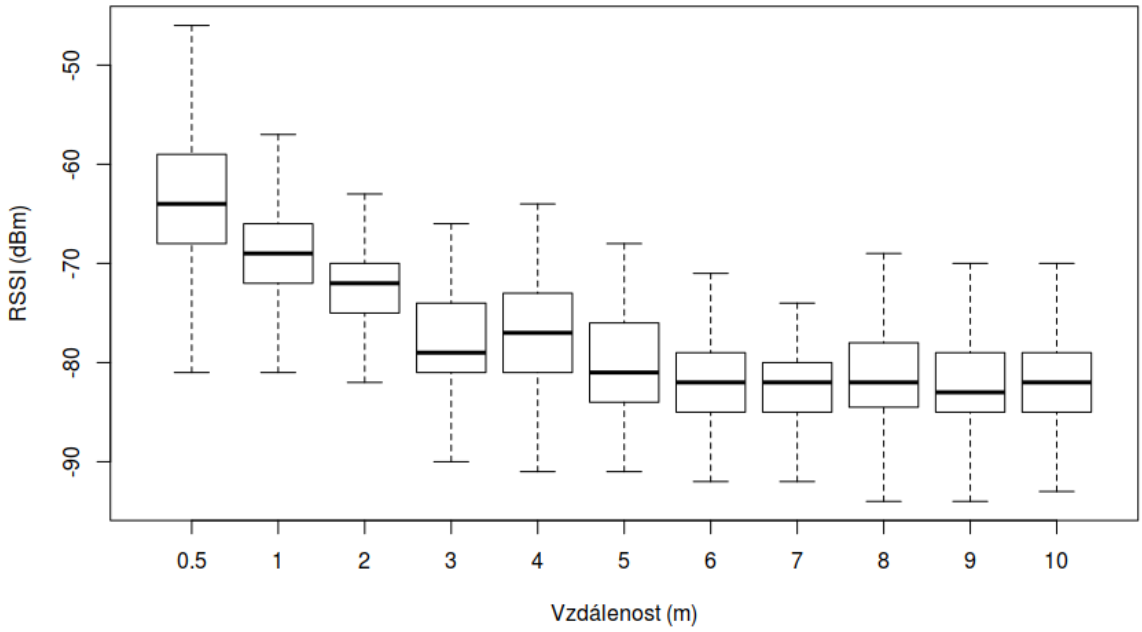
[15] RHUDY M. B., SALGUERO R. A. a HOLAPPA K. *A Kalman Filtering Tutorial For Undergraduate Students* [online]. Division of Engineering, Pennsylvania State University Dostupné z: <http://aircconline.com/ijcses/V8N1/8117ijcses01.pdf>

[16] WELCH G. a BISHOP G. *An Introduction to the Kalman Filter* [online]. Department of Computer Science, University of North Carolina at Chapel Hill Dostupné z: <http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf>

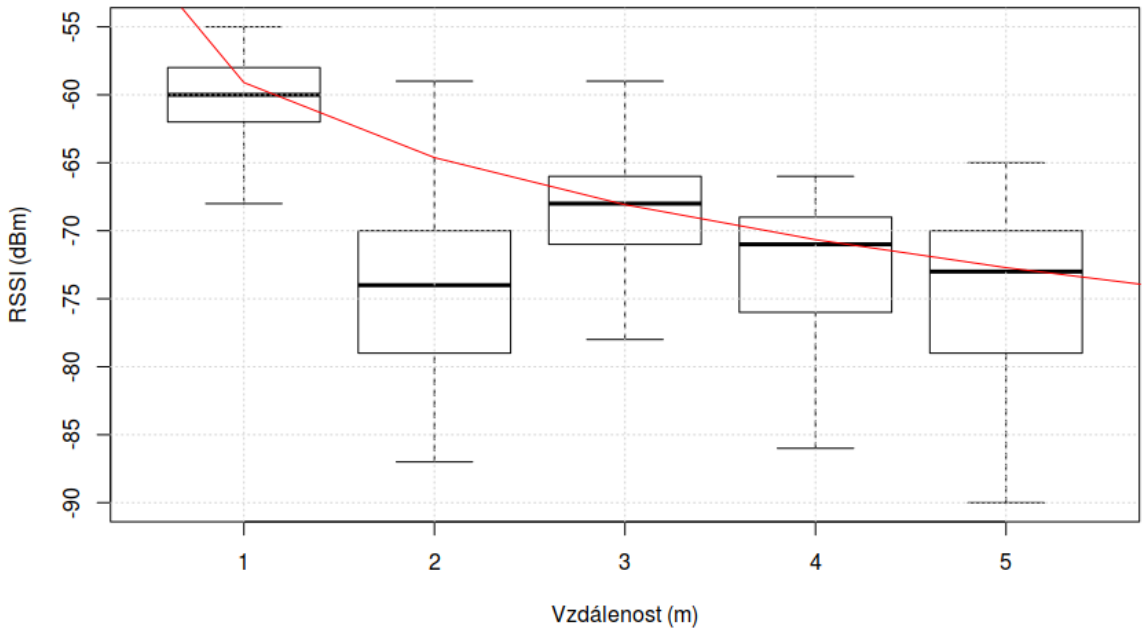
A Naměřené hodnoty RSSI podle vzdálenosti ve venkovním i vnitřním prostředí



Obrázek 22: Naměřené hodnoty RSSI podle vzdálenosti, kde měřící i vysílací stanice jsou orientovány stejným směrem a měřící stanice po boku od vysílací stanice. Odlehlá pozorování byla odstraněna.



Obrázek 23: Naměřené hodnoty RSSI podle vzdálenosti s útlumem lidské osoby. Odlehlá pozorování byla odstraněna.



Obrázek 24: Naměřené hodnoty RSSI podle vzdálenosti na přímou vzdálenost ve vnitřních prostorách. Červená křivka reprezentuje funkci 7. Odlehlá pozorování byla odstraněna.

B Seznam použitých knihoven a nástrojů

B.1 Knihovny

- Android SDK & Support - https://developer.android.com
- Android-GeoGson - https://github.com/3sidedcube/Android-GeoGson
- Android-SpinKit - https://github.com/ybq/Android-SpinKit
- Babel - https://babeljs.io/
- Brunch - https://brunch.io/
- EventBus - http://greenrobot.org/eventbus/
- ESP-IDF - https://github.com/espressif/esp-idf
- Deeplearning4j - https://deeplearning4j.org/
- Flow - https://github.com/facebook/flow
- Gradle - https://gradle.org/
- gson - https://github.com/google/gson
- jackson - https://github.com/FasterXML/jackson
- JGraphT - http://jgrapht.org/
- Kotlin - https://kotlinlang.org/
- Kotlin - Serializable - https://github.com/Kotlin/kotlinx.serialization
- Leaflet - http://leafletjs.com/
- Leaflet Indoor - https://github.com/cbaines/leaflet-indoor
- Leaflet Ant Path - https://github.com/rubenspgcavalcante/leaflet-ant-path
- Leaflet Plugins - https://github.com/shramov/leaflet-plugins
- Material Icons - https://material.io/icons/
- MaterialSearchBar - https://github.com/mancj/MaterialSearchBar
- Math – Commons Math - http://commons.apache.org/proper/commons-math/
- RxAndroidBle - https://github.com/Polidea/RxAndroidBle

- RxKotlin - https://github.com/ReactiveX/RxKotlin
- ReactiveSensors - https://github.com/pwittchen/ReactiveSensors
- Sensor Fusion - https://github.com/apacha/sensor-fusion-demo/
- Trilateration - https://github.com/lemmingapex/Trilateration
- The JTS Topology Suite - https://github.com/locationtech/jts
- 137-geo - https://github.com/lukehb/137-geo

B.1.1 Další

- Ikona aplikace - https://www.flaticon.com/free-icon/placeholder_285057

B.2 Nástroje

- Android Studio - https://developer.android.com/studio/index.html
- IntelliJ IDEA - https://www.jetbrains.com/idea/
- JOSM - https://josm.openstreetmap.de/
- npm - https://www.npmjs.com/
- RStudio - https://www.rstudio.com/
- WebStorm - https://www.jetbrains.com/webstorm/

C Příloha CD

Obsah složek:

- `zdrojove kody` Zdrojové kódy
  - `ble-adv` zdrojové kódy majáčku
  - `desktop-geoprojection` zobrazovací nástroj logů
  - `geoprojection` lokalizační a vyhledávací algoritmy (společná knihovna)
  - `leguide` zdrojové kódy Android aplikace
  - `train-network` zdrojové kódy pro učení neuronové sítě
  - `measuring` aplikace pro měření signálu *Measuring BLE*
  - `web-map` zdrojové kódy webové aplikace
- `mapove podklady` Mapové podklady
  - `rodinny dum` Zmapovaný rodinný dům
  - `vsb` Zmapované čtvrté podlaží budovy FEI
- `namerene hodnoty` Naměřené hodnoty pro tvorbu křivky včetně R souboru pro analýzu

D Návod ke zprovoznění práce

Pro tvorbu majáčku je potřeba mít nainstalované `esp-idf`. Následně ve složce `zdrojove kody/ble-adv` stačí spustit příkaz `make flash`, který zdrojové kódy zkompiluje a nahraje do modulu ESP32.

Mobilní aplikace se skládá z webové části. Tato část je ve složce `zdrojove kody/web-map`. Pro vygenerování nové verze s úpravou je potřeba mít nainstalovaný nástroj `npm`. Následně stačí spustit `npm install` a `npm start` v této složce, kdy se vytvoří webový server. Ten následně stačí ukončit a v podsložce `public` se nacházejí soubory, které stačí zabalit do zipu (do kořene zipu) s názvem `page.zip`. Tento zip je následně potřeba vložit do složky `zdrojove kody/leguide/app/src/main/assets`. V této složce se taktéž nacházejí mapové podklady ve formátu GeoJSON.

Mobilní aplikaci lze spustit standardními nástroji v Android Studio.